

Package: STPGA (via r-universe)

October 14, 2026

Type Package

Title Selection of Training Populations by Genetic Algorithm

Version 7.0.2

Date 2026-05-28

Author Deniz Akdemir

Maintainer Deniz Akdemir <deniz.akdemir.work@gmail.com>

Description Advanced genetic algorithms for optimal subset selection in high-dimensional prediction problems. Provides efficient single and multi-objective optimization for training population selection with comprehensive criteria including A-, D-, E-optimality, prediction error variance (PEV), Cook's distance (CD), and distance-based measures. Features multi-criteria convergence detection, restart mechanisms, rank-based selection with pressure control, adaptive mutation, diversity preservation, and numerically stable matrix operations. Includes convergence diagnostics, configurable optimization windows, and both modern clean interfaces with legacy compatibility functions.

License GPL-3

Depends R (>= 3.5.0), AlgDesign, scales, scatterplot3d, emoa, grDevices, parallel

Suggests EMMREML, quadprog, glmnet, leaps, Matrix, testthat (>= 3.0.0), knitr, rmarkdown, digest, progress

VignetteBuilder knitr

NeedsCompilation no

Encoding UTF-8

RoxygenNote 7.3.1

Repository <https://denizakdemir.r-universe.dev>

Date/Publication 2026-05-28 08:30:43 UTC

RemoteUrl <https://github.com/denizakdemir/stpga>

RemoteRef HEAD

RemoteSha d307a34778c6acc82f101ddc54e49288df77ac9f

Contents

a_optimality	4
adaptive_mutation_rate	5
block_matrix_mult	6
cache_stats	7
calculate_crowding_distance	7
calculate_diversity	8
cd_mean	8
cd_mean_mm	9
cd_mean_mm_h2	11
check_convergence	11
check_dominance	12
check_matrix_dimensions	13
check_memory_usage	13
check_multi_criteria_convergence	14
check_parameter_naming	14
chunked_crossprod	15
chunked_matrix_operation	15
clear_cache	16
compute_amatrix	16
compute_convergence_metrics	17
compute_generation_stats	17
compute_population_stats	18
compute_prediction_core	18
convergence_diagnostics	19
create_evaluation_cache	20
create_function_alias	21
create_progress_bar	21
criterion	22
crossover	23
crowding_replacement	24
d_optimality	24
determine_optimal_batch_size	26
determine_primary_matrix	26
distance_criterion	27
distance_internal_mean	28
distance_internal_min	28
distance_to_ideal	29
distance_train_to_test_max	30
distance_train_to_test_mean	30
diversity_preserving_selection	31
diversity_summary	32
document_parameter	32
e_optimality	33
elite_selection	34
environmental_selection	35
evaluate_multiobjective_population	35
evaluate_population	36

evaluate_population_smart	37
evaluate_solution_batch	38
extract_pareto_front	39
find_non_dominated	39
finish_progress	40
fitness_sharing	40
g_optimality	41
generate_doc_template	42
generate_multiobjective_offspring	43
generate_offspring	43
genomic_relationship_matrix	44
get_adaptive_ridge	45
get_function_variants	46
get_optimal_cores	46
group_similar_solutions	47
h2_to_variances	47
i_optimality	48
influence_measure_legacy	49
initialize_population	50
legacy-compatibility	51
matrix_stability_check	53
multiobjective_selection	53
mutate_solution	54
non_dominated_sorting	54
parallel_apply	55
perform_restart	55
pev_max	56
pev_mean	57
pev_mean_mm	59
pev_mean_mm_h2	60
plot_pareto_progress	61
population_distances	62
rank_selection	62
ridge_regression_cv	64
safe_matrix_inverse	64
select_diverse_subset	65
simple_cache	65
standardize_name	66
standardize_parameters	66
subset_ga	67
subset_ga_multiobjective	70
subset_ga_multiobjective_single	72
subset_ga_single	73
tournament_selection	74
transform_fitness	74
update_pareto_archive	75
update_progress	75
validate_kinship_matrix	76

validate_matrix_params	76
validate_selection_parameters	77
validate_variance_components	78
WheatData	79
Index	80

<code>a_optimality</code>	<i>A-optimality criterion for experimental design</i>
---------------------------	---

Description

Computes the A-optimality criterion, which minimizes the trace of the inverse covariance matrix. Lower values indicate better designs.

Usage

```
a_optimality(train, test = NULL, P, lambda = 1e-06, C = NULL)
```

Arguments

<code>train</code>	Vector of training set individual names
<code>test</code>	Vector of test set individual names (if NULL, uses training set)
<code>P</code>	Prediction matrix with individuals as rows and variables as columns
<code>lambda</code>	Ridge regularization parameter for numerical stability (default: 1e-6)
<code>C</code>	Contrast matrix for specific linear combinations (default: NULL)

Details

A-optimality minimizes the average prediction variance. It is equivalent to minimizing the trace of $(X'X)^{-1}$ where X is the design matrix.

Mathematical formula: $A = \text{tr}((X'X)^{-1})$

Value

A-optimality value (trace of inverse covariance matrix)

References

Fedorov, V.V. (1972). Theory of Optimal Experiments. Academic Press.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for faster computation
set.seed(123)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:20]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define candidate and test sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute A-optimality (minimizes average prediction variance)
aopt_value <- a_optimality(train_set, test_set, PC_subset)
print(paste("A-optimality value:", round(aopt_value, 6)))

# Compare different training set sizes
small_train <- sample(candidates, 10)
large_train <- sample(candidates, 30)

aopt_small <- a_optimality(small_train, test_set, PC_subset)
aopt_large <- a_optimality(large_train, test_set, PC_subset)

print(paste("Small training (n=10):", round(aopt_small, 6)))
print(paste("Large training (n=30):", round(aopt_large, 6)))
print(paste("Improvement factor:", round(aopt_small / aopt_large, 2)))
```

adaptive_mutation_rate

Adaptive mutation rate based on population diversity

Description

Adaptive mutation rate based on population diversity

Usage

```
adaptive_mutation_rate(
  base_mutprob,
  diversity,
```

```

    generation,
    max_generations,
    min_rate = 0.01,
    max_rate = 0.95
)

```

Arguments

<code>base_mutprob</code>	Base mutation probability
<code>diversity</code>	Current population diversity (0-1)
<code>generation</code>	Current generation number
<code>max_generations</code>	Maximum generations
<code>min_rate</code>	Minimum mutation rate (default: 0.01)
<code>max_rate</code>	Maximum mutation rate (default: 0.95)

Value

Adjusted mutation probability

<code>block_matrix_mult</code>	<i>Efficient block matrix multiplication</i>
--------------------------------	--

Description

Efficient block matrix multiplication

Usage

```
block_matrix_mult(X, Y, block_size = 500)
```

Arguments

<code>X</code>	First matrix
<code>Y</code>	Second matrix
<code>block_size</code>	Block size for computation (default: 500)

Value

Matrix product X

cache_stats	<i>Get cache statistics</i>
-------------	-----------------------------

Description

Get cache statistics

Usage

```
cache_stats(cache_env = NULL)
```

Arguments

cache_env	Cache environment
-----------	-------------------

Value

List with cache statistics

calculate_crowding_distance	<i>Calculate crowding distance for diversity preservation</i>
-----------------------------	---

Description

Calculate crowding distance for diversity preservation

Usage

```
calculate_crowding_distance(fitness)
```

Arguments

fitness	Fitness matrix for a single front
---------	-----------------------------------

Value

Vector of crowding distances

calculate_diversity	<i>Calculate population diversity</i>
---------------------	---------------------------------------

Description

Calculate population diversity

Usage

```
calculate_diversity(population)
```

Arguments

population	List of solutions in the population
------------	-------------------------------------

Value

Diversity measure (0 to 1, higher = more diverse)

cd_mean	<i>Coefficient of Determination (R^2) based on experimental design literature</i>
---------	--

Description

Computes the coefficient of determination as the proportion of variance explained by the model, using the hat matrix diagonal (leverage values).

Usage

```
cd_mean(train, test = NULL, P, lambda = 1e-06, C = NULL, normalized = FALSE)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
normalized	Whether to normalize by trace for scale-invariance (default: FALSE)

Details

Computes R^2 as the average leverage from the hat matrix, which represents the proportion of variance explained by the model in experimental design.

Formula: $R^2 = \text{tr}(H)/n$ where $H = X(X'X)^{-1}X'$ is the hat matrix

This is the standard definition used when response data is not available.

Value

Mean R^2 value (proportion of variance explained)

References

Fedorov, V.V. (1972). Theory of Optimal Experiments. Academic Press. Atkinson, A.C., Donev, A.N., Tobias, R.D. (2007). Optimum Experimental Designs.

cd_mean_mm	<i>Coefficient of Determination (R^2) for Mixed Models</i>
------------	---

Description

Computes the coefficient of determination for mixed models, measuring the expected prediction accuracy of random effects (breeding values).

Usage

```
cd_mean_mm(train, test = NULL, P, K, lambda = 1e-06, C = NULL, Vg = 1, Ve = 1)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and markers as columns
K	Kinship/relationship matrix (symmetric, positive definite)
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
Vg	Genetic variance component (default: 1)
Ve	Error variance component (default: 1)

Details

The Coefficient of Determination for BLUP of random effects is defined as: $R^2 = 1 - \text{PEV}/\sigma_u^2$

This represents the proportion of genetic variance that can be predicted, or the reliability of breeding value prediction.

Value

Mean R^2 value for mixed models (higher indicates better prediction accuracy)

References

Henderson, C.R. (1984). Applications of Linear Models in Animal Breeding.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for mixed model demonstration
set.seed(159)
subset_indices <- sample(1:nrow(Wheat.M), 60)
M_subset <- Wheat.M[subset_indices, 1:50]
K_subset <- Wheat.K[subset_indices, subset_indices]

# Define train and test sets
all_individuals <- rownames(M_subset)
test_set <- sample(all_individuals, 12)
train_set <- sample(setdiff(all_individuals, test_set), 18)

# Mixed model R2 (reliability) with different heritability scenarios
h2_values <- c(0.2, 0.5, 0.8)

print("R2 (Coefficient of Determination) - Different Heritabilities:")
for (h2 in h2_values) {
  var_comps <- h2_to_variances(h2)
  cd_mm <- cd_mean_mm(train_set, test_set, M_subset, K_subset,
    Vg = var_comps$Vg, Ve = var_comps$Ve)
  print(paste("h2 =", h2, "-> R2 =", round(cd_mm, 4)))
}

# Compare training set sizes
small_train <- sample(setdiff(all_individuals, test_set), 10)
large_train <- sample(setdiff(all_individuals, test_set), 25)

var_comps <- h2_to_variances(0.5) # Medium heritability
cd_small <- cd_mean_mm(small_train, test_set, M_subset, K_subset,
  Vg = var_comps$Vg, Ve = var_comps$Ve)
cd_large <- cd_mean_mm(large_train, test_set, M_subset, K_subset,
  Vg = var_comps$Vg, Ve = var_comps$Ve)

print(paste("Small training set (n=10): R2 =", round(cd_small, 4)))
print(paste("Large training set (n=25): R2 =", round(cd_large, 4)))
print(paste("Improvement with larger set:", round(cd_large - cd_small, 4)))
```

cd_mean_mm_h2	<i>Coefficient of Determination for Mixed Models using heritability</i>
---------------	---

Description

Coefficient of Determination for Mixed Models using heritability

Usage

```
cd_mean_mm_h2(
  train,
  test = NULL,
  P,
  K,
  h2,
  lambda = 1e-06,
  C = NULL,
  total_var = 1
)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and markers as columns
K	Kinship/relationship matrix (symmetric, positive definite)
h2	Heritability (between 0 and 1)
lambda	Ridge regularization parameter (default: 1e-6)
C	Contrast matrix (default: NULL)
total_var	Total variance (default: 1)

Value

Mean R^2 value

check_convergence	<i>Check convergence based on convergence history</i>
-------------------	---

Description

Check convergence based on convergence history

Usage

```
check_convergence(  
  convergence_history,  
  current_generation,  
  window_size = 20,  
  tolerance = 1e-06  
)
```

Arguments

- convergence_history Matrix of convergence metrics over generations
- current_generation Current generation number
- window_size Window size for checking convergence (default: 20)
- tolerance Convergence tolerance (default: 1e-6)

Value

TRUE if converged, FALSE otherwise

check_dominance	<i>Check dominance between two solutions</i>
-----------------	--

Description

Check dominance between two solutions

Usage

```
check_dominance(solution1, solution2, criteria_types)
```

Arguments

- solution1 First solution fitness
- solution2 Second solution fitness
- criteria_types Optimization types

Value

1 if solution1 dominates, -1 if solution2 dominates, 0 if non-dominated

```
check_matrix_dimensions
```

Check matrix dimensions for operations

Description

Check matrix dimensions for operations

Usage

```
check_matrix_dimensions(...)
```

Arguments

... Named matrices to check

Value

NULL if dimensions are compatible, throws error otherwise

```
check_memory_usage
```

Check memory usage and warn for large computations

Description

Check memory usage and warn for large computations

Usage

```
check_memory_usage(n_individuals, n_variables, operation = "matrix_mult")
```

Arguments

n_individuals Number of individuals

n_variables Number of variables/markers

operation Type of operation ("matrix_mult", "eigen", "svd", "chol")

Value

NULL, but issues warnings for large computations

`check_multi_criteria_convergence`

Multi-criteria convergence detection

Description

Multi-criteria convergence detection

Usage

```
check_multi_criteria_convergence(
  convergence_history,
  generation,
  window_size,
  tolerance
)
```

Arguments

<code>convergence_history</code>	List containing convergence metrics history
<code>generation</code>	Current generation number
<code>window_size</code>	Window size for convergence check
<code>tolerance</code>	Convergence tolerance

Value

List with convergence status and reason

`check_parameter_naming`

Check if parameter names follow conventions

Description

Check if parameter names follow conventions

Usage

```
check_parameter_naming(param_names)
```

Arguments

<code>param_names</code>	Vector of parameter names
--------------------------	---------------------------

Value

List with standardized names and warnings

chunked_crossprod	<i>Memory-efficient crossprod computation</i>
-------------------	---

Description

Memory-efficient crossprod computation

Usage

```
chunked_crossprod(X, Y = NULL, chunk_size = 1000)
```

Arguments

X	Matrix
Y	Optional second matrix (default: X)
chunk_size	Chunk size for computation (default: 1000)

Value

t(X)

chunked_matrix_operation	<i>Compute matrix operations in chunks for memory efficiency</i>
--------------------------	--

Description

Compute matrix operations in chunks for memory efficiency

Usage

```
chunked_matrix_operation(X, FUN, chunk_size = 1000, ...)
```

Arguments

X	Matrix to process
FUN	Function to apply to each chunk
chunk_size	Maximum chunk size (default: 1000)
...	Additional arguments to FUN

Value

Combined result from all chunks

<code>clear_cache</code>	<i>Clear computation cache</i>
--------------------------	--------------------------------

Description

Clear computation cache

Usage

```
clear_cache(cache_env = NULL)
```

Arguments

<code>cache_env</code>	Cache environment to clear
------------------------	----------------------------

<code>compute_amatrix</code>	<i>STPGA Matrix Operations</i>
------------------------------	--------------------------------

Description

Matrix utilities for genetic relationship computations

Usage

```
compute_amatrix(M, pieces = 10, mc.cores = 1)
```

Arguments

<code>M</code>	Marker matrix coded as -1,0,1 (individuals x markers)
<code>pieces</code>	Number of pieces to split computation (default: 10)
<code>mc.cores</code>	Number of cores for parallel processing (default: 1)

Value

Combined genomic relationship matrix (A matrix)

Author(s)

Deniz Akdemir Compute A matrix in pieces for large marker datasets

`compute_convergence_metrics`*Compute convergence metrics for multi-objective optimization*

Description

Compute convergence metrics for multi-objective optimization

Usage

```
compute_convergence_metrics(fitness)
```

Arguments

`fitness` Archive fitness matrix

Value

Vector of convergence metrics

`compute_generation_stats`*Compute generation statistics for multi-objective optimization*

Description

Compute generation statistics for multi-objective optimization

Usage

```
compute_generation_stats(population, fitness, archive_fitness)
```

Arguments

`population` Current population
`fitness` Current fitness matrix
`archive_fitness` Archive fitness matrix

Value

List with generation statistics

```
compute_population_stats
```

Population statistics computation

Description

Population statistics computation

Usage

```
compute_population_stats(population, fitness)
```

Arguments

<code>population</code>	List of solutions
<code>fitness</code>	Vector of fitness values

Value

List with population statistics

```
compute_prediction_core
```

Compute prediction core for matrix operations

Description

This function computes the core matrix operations used by all optimization criteria. It handles numerical stability and implements the correct PEV formula from literature.

Usage

```
compute_prediction_core(
  p_train,
  p_test = NULL,
  lambda = 1e-06,
  C = NULL,
  adaptive_ridge = TRUE
)
```

Arguments

<code>p_train</code>	Training prediction matrix (n_train x p)
<code>p_test</code>	Test prediction matrix (n_test x p). If NULL, uses training set
<code>lambda</code>	Ridge regularization parameter for numerical stability (default: 1e-6)
<code>C</code>	Contrast matrix for specific linear combinations (default: NULL)
<code>adaptive_ridge</code>	Whether to use adaptive ridge parameter (default: TRUE)

Value

List containing core matrices and computations

`convergence_diagnostics`

Get convergence history diagnostics from GA results

Description

Get convergence history diagnostics from GA results

Usage

```
convergence_diagnostics(ga_result, plot = TRUE)
```

Arguments

<code>ga_result</code>	Result from <code>subset_ga</code> function
<code>plot</code>	Whether to create diagnostic plots (default: TRUE)

Value

List with convergence diagnostics

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(456)
subset_indices <- sample(1:nrow(Wheat.M), 100)
M_subset <- Wheat.M[subset_indices, 1:30]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define candidate and test sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 20)
candidates <- setdiff(all_individuals, test_set)

# Run genetic algorithm with convergence tracking
ga_result <- subset_ga(
  P = PC_subset,
  Candidates = candidates,
  Test = test_set,
```

```

    ntoselect = 15,
    npop = 25,
    niterations = 40,
    criterion = "pev_mean",
    enable_restart = TRUE,
    convergence_window_multiplier = 4,
    verbose = FALSE
)

# Generate convergence diagnostics
diags <- convergence_diagnostics(ga_result, plot = FALSE)

print("Convergence Diagnostics:")
print(paste("Converged:", diags$convergence_status$converged))
print(paste("Reason:", diags$convergence_status$reason))
print(paste("Window size used:", diags$window_size))
print(paste("Data points analyzed:", diags$data_points))

print("Stability Metrics:")
print(paste("Fitness stability:", round(diags$metrics$fitness_stability, 8)))
print(paste("Diversity stability:", round(diags$metrics$diversity_stability, 6)))
print(paste("Mean improvement rate:", round(diags$metrics$mean_improvement_rate, 8)))

if (length(diags$recommendations) > 0) {
  print("Recommendations:")
  for (rec in diags$recommendations) {
    print(paste("-", rec))
  }
}

```

```
create_evaluation_cache
```

Create evaluation cache for storing computed fitness values

Description

Create evaluation cache for storing computed fitness values

Usage

```
create_evaluation_cache()
```

Value

Cache object with get and store methods

`create_function_alias`*Create function alias for backward compatibility*

Description

Create function alias for backward compatibility

Usage

```
create_function_alias(old_name, new_name, envir = .GlobalEnv)
```

Arguments

<code>old_name</code>	Old function name
<code>new_name</code>	New standardized function name
<code>envir</code>	Environment to create alias in

`create_progress_bar` *Performance Optimization Functions*

Description

Functions to improve computational performance for large datasets

Usage

```
create_progress_bar(total, title = "Processing", verbose = TRUE)
```

Arguments

<code>total</code>	Total number of iterations
<code>title</code>	Title for the progress bar
<code>verbose</code>	Whether to show progress bar

Value

Progress bar object or NULL

Author(s)

Deniz Akdemir Add progress bar for long-running computations

criterion	<i>Unified criterion function for optimization</i>
-----------	--

Description

A unified interface to all optimization criteria. This function automatically dispatches to the appropriate criterion based on the criterion name and parameters.

Usage

```
criterion(
  train,
  test = NULL,
  P,
  lambda = 1e-06,
  C = NULL,
  criterion = "pev_mean",
  K = NULL,
  Vg = NULL,
  Ve = NULL
)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
criterion	Character string specifying the criterion to compute
K	Kinship matrix for mixed models (required for mixed model criteria)
Vg	Genetic variance component for mixed models (default: 1 if K provided)
Ve	Error variance component for mixed models (default: 1 if K provided)

Details

Available criteria:

****Classical Optimality:**** - "a_optimality": A-optimality (trace of inverse covariance) - "d_optimality": D-optimality (log determinant) - "e_optimality": E-optimality (minimum eigenvalue) - "g_optimality": G-optimality (maximum prediction variance) - "i_optimality": I-optimality (average prediction variance)

****Prediction-Based:**** - "pev_mean": Mean prediction error variance - "pev_mean_normalized": Normalized mean PEV - "pev_max": Maximum prediction error variance - "pev_max_normalized": Normalized maximum PEV - "cd_mean": Coefficient of determination (R^2) - "cd_mean_normalized": Normalized R^2

****Mixed Models:**** - "pev_mean_mm": Mean PEV for mixed models (requires K) - "cd_mean_mm": R² for mixed models (requires K)

****Legacy (backward compatibility):**** - "AOPT", "DOPT", "EOPT": Classical optimality criteria - "PEVMEAN", "PEVMEAN2": PEV (normalized version) - "PEVMAX", "PEVMAX2": Maximum PEV (normalized version) - "CDMEAN", "CDMEAN2": R² (normalized version) - "PEVMEANMM", "CDMEANMM": Mixed model versions

Value

Criterion value (interpretation depends on specific criterion)

crossover

STPGA Genetic Algorithm Operators

Description

Crossover, mutation, and selection operators for genetic algorithms

Usage

```
crossover(
  parent1,
  parent2,
  ntoselect,
  candidates,
  method = "uniform",
  bias = 0.5
)
```

Arguments

parent1	First parent solution
parent2	Second parent solution
ntoselect	Target size of offspring
candidates	Pool of candidate individuals
method	Crossover method: "uniform", "twopoint", "proportional" (default: "uniform")
bias	Bias toward parent1 (0.5 = equal, >0.5 favors parent1) (default: 0.5)

Value

Offspring solution

Author(s)

Deniz Akdemir Crossover operation for subset selection

`crowding_replacement` *Crowding replacement for diversity preservation*

Description

Crowding replacement for diversity preservation

Usage

```
crowding_replacement(  
  offspring,  
  offspring_fitness,  
  population,  
  population_fitness,  
  crowding_factor = 3  
)
```

Arguments

`offspring` New offspring solutions
`offspring_fitness` Fitness of offspring
`population` Current population
`population_fitness` Current population fitness
`crowding_factor` Number of candidates to compare for replacement

Value

Updated population after crowding replacement

`d_optimality` *D-optimality criterion for experimental design*

Description

Computes the D-optimality criterion, which minimizes the determinant of the inverse covariance matrix. Lower values indicate better designs.

Usage

```
d_optimality(train, test = NULL, P, lambda = 1e-06, C = NULL)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)

Details

D-optimality minimizes the generalized variance of parameter estimates. It is equivalent to maximizing the determinant of $X'X$ where X is the design matrix.

Mathematical formula: $D = \log(\det(X'X))$

Value

D-optimality value (negative log determinant)

References

Kiefer, J. (1959). Optimum experimental designs. *Journal of the Royal Statistical Society B*.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(456)
subset_indices <- sample(1:nrow(Wheat.M), 60)
M_subset <- Wheat.M[subset_indices, 1:15]

# Extract principal components for D-optimality
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:4]
rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 10)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 15)

# Compute D-optimality (minimizes generalized variance)
dopt_value <- d_optimality(train_set, test_set, PC_subset)
print(paste("D-optimality value:", round(dopt_value, 4)))

# Compare with different training sets
train_set2 <- sample(candidates, 15)
dopt_value2 <- d_optimality(train_set2, test_set, PC_subset)
```

```

print(paste("Training set 1 D-opt:", round(dopt_value, 4)))
print(paste("Training set 2 D-opt:", round(dopt_value2, 4)))

if (dopt_value < dopt_value2) {
  print("Training set 1 is better (lower D-optimality)")
} else {
  print("Training set 2 is better (lower D-optimality)")
}

```

```
determine_optimal_batch_size
```

Determine optimal batch size for evaluation

Description

Determine optimal batch size for evaluation

Usage

```
determine_optimal_batch_size(n_solutions, mc.cores)
```

Arguments

<code>n_solutions</code>	Number of solutions
<code>mc.cores</code>	Number of cores

Value

Optimal batch size

```
determine_primary_matrix
```

Determine primary matrix for optimization

Description

Determine primary matrix for optimization

Usage

```
determine_primary_matrix(Pcs, Dist, Kernel)
```

Arguments

<code>Pcs</code>	Principal components matrix
<code>Dist</code>	Distance matrix
<code>Kernel</code>	Kernel matrix

Value

Primary matrix to use for optimization

<code>distance_criterion</code>	<i>Unified distance criterion function</i>
---------------------------------	--

Description

Unified distance criterion function

Usage

```
distance_criterion(  
  train,  
  test,  
  distance_matrix,  
  criterion = c("max_to_test", "mean_to_test", "neg_min_internal", "neg_mean_internal"),  
  lambda = NULL,  
  C = NULL  
)
```

Arguments

<code>train</code>	Training set individual names
<code>test</code>	Test set individual names
<code>distance_matrix</code>	Distance matrix
<code>criterion</code>	Type of distance criterion
<code>lambda</code>	Ridge parameter (for compatibility)
<code>C</code>	Contrast matrix (for compatibility)

Value

Distance criterion value

`distance_internal_mean`

Negative mean distance within training set

Description

Negative mean distance within training set

Usage

```
distance_internal_mean(
  train,
  test = NULL,
  distance_matrix,
  lambda = NULL,
  C = NULL
)
```

Arguments

<code>train</code>	Training set individual names
<code>test</code>	Test set individual names (unused, kept for compatibility)
<code>distance_matrix</code>	Distance matrix
<code>lambda</code>	Ridge parameter (unused, kept for compatibility)
<code>C</code>	Contrast matrix (unused, kept for compatibility)

Value

Negative mean internal distance

`distance_internal_min`

Negative minimum distance within training set

Description

Negative minimum distance within training set

Usage

```
distance_internal_min(
  train,
  test = NULL,
  distance_matrix,
  lambda = NULL,
  C = NULL
)
```

Arguments

train	Training set individual names
test	Test set individual names (unused, kept for compatibility)
distance_matrix	Distance matrix
lambda	Ridge parameter (unused, kept for compatibility)
C	Contrast matrix (unused, kept for compatibility)

Value

Negative minimum internal distance

distance_to_ideal	<i>STPGA Distance Functions</i>
-------------------	---------------------------------

Description

Distance-based criteria and utilities for subset selection

Usage

```
distance_to_ideal(X, method = "euclidean", handle_zeros = "warning")
```

Arguments

X	Matrix of solutions (rows) by objectives (columns)
method	Distance method: "euclidean", "manhattan", "chebyshev" (default: "euclidean")
handle_zeros	How to handle zero ranges: "warning", "error", "ignore" (default: "warning")

Value

Vector of distances to ideal point (0,0,...,0)

Author(s)

Deniz Akdemir Distance to ideal point calculation

`distance_train_to_test_max`*Maximum distance from training to test set*

Description

Maximum distance from training to test set

Usage

```
distance_train_to_test_max(  
  train,  
  test,  
  distance_matrix,  
  lambda = NULL,  
  C = NULL  
)
```

Arguments

<code>train</code>	Training set individual names
<code>test</code>	Test set individual names
<code>distance_matrix</code>	Distance matrix
<code>lambda</code>	Ridge parameter (unused, kept for compatibility)
<code>C</code>	Contrast matrix (unused, kept for compatibility)

Value

Maximum distance value

`distance_train_to_test_mean`*Mean distance from training to test set*

Description

Mean distance from training to test set

Usage

```
distance_train_to_test_mean(  
  train,  
  test,  
  distance_matrix,  
  lambda = NULL,  
  C = NULL  
)
```

Arguments

<code>train</code>	Training set individual names
<code>test</code>	Test set individual names
<code>distance_matrix</code>	Distance matrix
<code>lambda</code>	Ridge parameter (unused, kept for compatibility)
<code>C</code>	Contrast matrix (unused, kept for compatibility)

Value

Mean distance value

`diversity_preserving_selection`

Diversity-preserving selection for environmental selection

Description

Diversity-preserving selection for environmental selection

Usage

```
diversity_preserving_selection(
  population,
  fitness,
  target_size,
  criteria_types
)
```

Arguments

<code>population</code>	Combined population
<code>fitness</code>	Combined fitness matrix
<code>target_size</code>	Target population size
<code>criteria_types</code>	Optimization types

Value

Indices of selected solutions

diversity_summary	<i>Population diversity summary</i>
-------------------	-------------------------------------

Description

Population diversity summary

Usage

```
diversity_summary(solutions, method = "jaccard")
```

Arguments

solutions	List of solutions
method	Distance method for calculation

Value

List with diversity statistics

document_parameter	<i>Generate consistent parameter documentation</i>
--------------------	--

Description

Generate consistent parameter documentation

Usage

```
document_parameter(param_name, param_type, default_value = NULL, description)
```

Arguments

param_name	Parameter name
param_type	Parameter type
default_value	Default value
description	Parameter description

Value

Formatted roxygen2 parameter documentation

e_optimality

E-optimality criterion for experimental design

Description

Computes the E-optimality criterion, which minimizes the maximum eigenvalue of the inverse covariance matrix. Lower values indicate better designs.

Usage

```
e_optimality(train, test = NULL, P, lambda = 1e-06, C = NULL)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)

Details

E-optimality minimizes the maximum variance of any linear combination of parameters. It is equivalent to maximizing the minimum eigenvalue of $X'X$ where X is the design matrix.

Mathematical formula: $E = \log(\text{max}((X'X)^{-1}))$

Value

E-optimality value (negative log of minimum eigenvalue)

References

Atkinson, A.C., Donev, A.N., Tobias, R.D. (2007). Optimum Experimental Designs. Oxford.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(321)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:20]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
```

```

rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute E-optimality (minimizes maximum variance)
eopt_value <- e_optimality(train_set, test_set, PC_subset)
print(paste("E-optimality value:", round(eopt_value, 6)))

# Compare with A and D optimality for same training set
aopt_value <- a_optimality(train_set, test_set, PC_subset)
dopt_value <- d_optimality(train_set, test_set, PC_subset)

print(paste("A-optimality:", round(aopt_value, 6)))
print(paste("D-optimality:", round(dopt_value, 6)))
print(paste("E-optimality:", round(eopt_value, 6)))

```

elite_selection

Elite selection

Description

Elite selection

Usage

```
elite_selection(population, fitness, n_select)
```

Arguments

population	List of solutions
fitness	Vector of fitness values
n_select	Number of elites to select

Value

Indices of selected elite individuals

`environmental_selection`*Environmental selection for NSGA-II*

Description

Environmental selection for NSGA-II

Usage

```
environmental_selection(fitness, ranks, target_size)
```

Arguments

<code>fitness</code>	Combined fitness matrix
<code>ranks</code>	Dominance ranks
<code>target_size</code>	Target population size

Value

Indices of selected solutions

`evaluate_multiobjective_population`*Evaluate population for multiple objectives*

Description

Evaluate population for multiple objectives

Usage

```
evaluate_multiobjective_population(  
  population,  
  P,  
  test,  
  criteria,  
  lambda,  
  C,  
  mc.cores  
)
```

Arguments

<code>population</code>	List of solutions
<code>P</code>	Primary matrix
<code>test</code>	Test set
<code>criteria</code>	Vector of criteria names
<code>lambda</code>	Ridge parameter
<code>C</code>	Contrast matrix
<code>mc.cores</code>	Number of cores

Value

Matrix of fitness values (solutions x criteria)

`evaluate_population` *STPGA Population Evaluation Functions*

Description

Efficient population evaluation and fitness computation

Usage

```
evaluate_population(  
  population,  
  P,  
  test = NULL,  
  criterion = "PEVMEAN2",  
  lambda = 1e-06,  
  C = NULL,  
  K = NULL,  
  Vg = NULL,  
  Ve = NULL,  
  mc.cores = 1,  
  use_cache = TRUE  
)
```

Arguments

<code>population</code>	List of candidate solutions (training sets)
<code>P</code>	Prediction matrix
<code>test</code>	Test set individual names (can be NULL)
<code>criterion</code>	Optimization criterion name (default: "PEVMEAN2")
<code>lambda</code>	Ridge regression parameter (default: 1e-6)
<code>C</code>	Contrast matrix (default: NULL)

<code>K</code>	Kinship matrix for mixed models (default: NULL)
<code>Vg</code>	Genetic variance matrix (default: NULL)
<code>Ve</code>	Error variance matrix (default: NULL)
<code>mc.cores</code>	Number of cores for parallel processing (default: 1)
<code>use_cache</code>	Whether to use computation cache (default: TRUE)

Value

Vector of fitness values for population

Author(s)

Deniz Akdemir Optimized population evaluation with smart caching

`evaluate_population_smart`

Smart population evaluation with adaptive batch processing

Description

Smart population evaluation with adaptive batch processing

Usage

```
evaluate_population_smart(
  population,
  P,
  test = NULL,
  criterion = "PEVMEAN2",
  lambda = 1e-06,
  C = NULL,
  K = NULL,
  Vg = NULL,
  Ve = NULL,
  mc.cores = 1,
  batch_size = NULL
)
```

Arguments

<code>population</code>	List of solutions
<code>P</code>	Prediction matrix
<code>test</code>	Test set names
<code>criterion</code>	Criterion name
<code>lambda</code>	Ridge parameter

C	Contrast matrix
K	Kinship matrix
Vg	Genetic variance
Ve	Error variance
mc.cores	Number of cores
batch_size	Batch size for processing (default: NULL for automatic)

Value

Vector of fitness values

evaluate_solution_batch

Evaluate a batch of solutions efficiently

Description

Evaluate a batch of solutions efficiently

Usage

```
evaluate_solution_batch(
  solutions,
  P,
  test,
  criterion,
  lambda,
  C,
  K,
  Vg,
  Ve,
  mc.cores
)
```

Arguments

solutions	List of solutions in the batch
P	Prediction matrix
test	Test set
criterion	Criterion name
lambda	Ridge parameter
C	Contrast matrix
K	Kinship matrix
Vg	Genetic variance
Ve	Error variance
mc.cores	Number of cores

Value

Vector of fitness values

<code>extract_pareto_front</code>	<i>Extract final Pareto front</i>
-----------------------------------	-----------------------------------

Description

Extract final Pareto front

Usage

```
extract_pareto_front(archive, fitness, criteria_types)
```

Arguments

<code>archive</code>	Archive of solutions
<code>fitness</code>	Archive fitness matrix
<code>criteria_types</code>	Optimization types

Value

List with Pareto solutions and fitness

<code>find_non_dominated</code>	<i>Find non-dominated solutions</i>
---------------------------------	-------------------------------------

Description

Find non-dominated solutions

Usage

```
find_non_dominated(fitness, criteria_types)
```

Arguments

<code>fitness</code>	Fitness matrix
<code>criteria_types</code>	Optimization types

Value

Indices of non-dominated solutions

<code>finish_progress</code>	<i>Finish progress bar</i>
------------------------------	----------------------------

Description

Finish progress bar

Usage

`finish_progress(pb)`

Arguments

`pb` Progress bar object

<code>fitness_sharing</code>	<i>Fitness sharing for diversity preservation</i>
------------------------------	---

Description

Fitness sharing for diversity preservation

Usage

`fitness_sharing(fitness, population, sharing_radius = 0.1, alpha = 1)`

Arguments

`fitness` Vector of fitness values
`population` List of solutions
`sharing_radius` Radius for fitness sharing (default: 0.1)
`alpha` Sharing function exponent (default: 1)

Value

Adjusted fitness values

<code>g_optimality</code>	<i>G-optimality criterion (minimizes maximum prediction variance)</i>
---------------------------	---

Description

G-optimality criterion (minimizes maximum prediction variance)

Usage

```
g_optimality(train, test = NULL, P, lambda = 1e-06)
```

Arguments

<code>train</code>	Vector of training set individual names
<code>test</code>	Vector of test set individual names (if NULL, uses training set)
<code>P</code>	Prediction matrix with individuals as rows and variables as columns
<code>lambda</code>	Ridge regularization parameter (default: 1e-6)

Details

G-optimality minimizes the maximum entry in the diagonal of the hat matrix, providing the best worst-case prediction precision.

Value

G-optimality value

References

Kiefer, J. (1975). Construction and optimality of generalized Youden designs.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(753)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:20]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
```

```

candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute G-optimality (minimizes maximum prediction variance)
gopt_value <- g_optimality(train_set, test_set, PC_subset)
print(paste("G-optimality value:", round(gopt_value, 6)))

# Compare with other optimality criteria
aopt <- a_optimality(train_set, test_set, PC_subset)
dopt <- d_optimality(train_set, test_set, PC_subset)
eopt <- e_optimality(train_set, test_set, PC_subset)

print("Optimality Comparison:")
print(paste("A-optimality (average variance):", round(aopt, 6)))
print(paste("D-optimality (volume):", round(dopt, 6)))
print(paste("E-optimality (max eigenvalue):", round(eopt, 6)))
print(paste("G-optimality (max hat diagonal):", round(gopt_value, 6)))

```

```
generate_doc_template
```

Generate standard function documentation template

Description

Generate standard function documentation template

Usage

```
generate_doc_template(func_name, purpose, category = "General")
```

Arguments

<code>func_name</code>	Function name
<code>purpose</code>	Function purpose
<code>category</code>	Function category

Value

Roxygen2 documentation template

```
generate_multiobjective_offspring
```

Generate offspring for multi-objective optimization

Description

Generate offspring for multi-objective optimization

Usage

```
generate_multiobjective_offspring(  
    parents,  
    candidates,  
    ntoselect,  
    n_offspring,  
    mutprob,  
    mutintensity,  
    mc.cores  
)
```

Arguments

<code>parents</code>	Parent solutions
<code>candidates</code>	Candidate pool
<code>ntoselect</code>	Solution size
<code>n_offspring</code>	Number of offspring to generate
<code>mutprob</code>	Mutation probability
<code>mutintensity</code>	Mutation intensity
<code>mc.cores</code>	Number of cores

Value

List of offspring solutions

```
generate_offspring
```

Generate offspring from elite solutions

Description

Generate offspring from elite solutions

Usage

```
generate_offspring(
  elites,
  candidates,
  npop,
  mutprob,
  mc.cores = 1,
  mutintensity = 1,
  tabu_memory = NULL,
  ntoselect = NULL,
  crossover_method = "uniform",
  parallel_threshold = 50
)
```

Arguments

<code>elites</code>	List of elite solutions
<code>candidates</code>	Pool of candidate individuals
<code>npop</code>	Population size to generate
<code>mutprob</code>	Mutation probability
<code>mc.cores</code>	Number of cores for parallel processing (default: 1)
<code>mutintensity</code>	Mutation intensity (default: 1)
<code>tabu_memory</code>	Tabu memory for avoiding revisits (default: NULL)
<code>ntoselect</code>	Target size of solutions (default: NULL, inferred from elites)
<code>crossover_method</code>	Method for crossover operations (default: "uniform")
<code>parallel_threshold</code>	Minimum population size for parallel processing (default: 50)

Value

List of offspring solutions

`genomic_relationship_matrix`

Compute genomic relationship matrix using different methods

Description

Compute genomic relationship matrix using different methods

Usage

```
genomic_relationship_matrix(M, method = "vanraden", pieces = 10, mc.cores = 1)
```

Arguments

M	Marker matrix coded as -1,0,1
method	Method for computation: "vanraden", "pieces", "standard" (default: "vanraden")
pieces	Number of pieces for piece-wise computation (default: 10)
mc.cores	Number of cores for parallel processing (default: 1)

Value

Genomic relationship matrix

<code>get_adaptive_ridge</code>	<i>Get adaptive ridge parameter based on condition number</i>
---------------------------------	---

Description

Get adaptive ridge parameter based on condition number

Usage

```
get_adaptive_ridge(
  X,
  target_condition = 1e+08,
  min_lambda = 1e-10,
  max_lambda = 0.01
)
```

Arguments

X	Matrix to analyze
target_condition	Target condition number (default: 1e8)
min_lambda	Minimum lambda value (default: 1e-10)
max_lambda	Maximum lambda value (default: 1e-2)

Value

Adaptive ridge parameter

```
get_function_variants
```

Get consistent function name variants

Description

Get consistent function name variants

Usage

```
get_function_variants(base_name)
```

Arguments

`base_name` Base function name

Value

List with name variants for different contexts

```
get_optimal_cores
```

Get optimal number of cores for computation

Description

Get optimal number of cores for computation

Usage

```
get_optimal_cores(data_size, operation_type = "mixed")
```

Arguments

`data_size` Approximate size of data
`operation_type` Type of operation ("cpu", "memory", "mixed")

Value

Recommended number of cores

group_similar_solutions

Group similar solutions for batch processing

Description

Group similar solutions for batch processing

Usage

```
group_similar_solutions(population, target_batch_size)
```

Arguments

population List of solutions
target_batch_size Target size for each batch

Value

List of solution groups

h2_to_variances

STPGA Optimization Criteria Functions

Description

Literature-corrected optimization criteria for subset selection in experimental design

Usage

```
h2_to_variances(h2, total_var = 1)
```

Arguments

h2 Heritability (proportion of variance due to genetics)
total_var Total variance (default: 1)

Value

List with Vg (genetic variance) and Ve (error variance)

Author(s)

Deniz Akdemir

References

Fedorov, V.V. (1972). Theory of Optimal Experiments. Academic Press. Henderson, C.R. (1984). Applications of Linear Models in Animal Breeding. Kiefer, J. (1959). Optimum experimental designs. Journal of the Royal Statistical Society B. Searle, S.R., Casella, G., McCulloch, C.E. (1992). Variance Components. Wiley. Atkinson, A.C., Donev, A.N., Tobias, R.D. (2007). Optimum Experimental Designs. Oxford. Convert between variance components and heritability

Examples

```
# Convert heritability to variance components
h2_values <- c(0.1, 0.3, 0.5, 0.7, 0.9)

for (h2 in h2_values) {
  var_comps <- h2_to_variances(h2)
  print(paste("h² =", h2, "-> Vg =", var_comps$Vg, ", Ve =", var_comps$Ve))
}

# Example with custom total variance
var_comps_scaled <- h2_to_variances(0.6, total_var = 2.5)
print(paste("h² = 0.6, total_var = 2.5 -> Vg =", var_comps_scaled$Vg,
            ", Ve =", var_comps_scaled$Ve))
```

<code>i_optimality</code>	<i>I-optimality criterion (minimizes average prediction variance)</i>
---------------------------	---

Description

I-optimality criterion (minimizes average prediction variance)

Usage

```
i_optimality(train, test = NULL, P, lambda = 1e-06)
```

Arguments

<code>train</code>	Vector of training set individual names
<code>test</code>	Vector of test set individual names (if NULL, uses training set)
<code>P</code>	Prediction matrix with individuals as rows and variables as columns
<code>lambda</code>	Ridge regularization parameter (default: 1e-6)

Details

I-optimality minimizes the average prediction variance over the design space.

Value

I-optimality value

References

Fedorov, V.V. (1972). Theory of Optimal Experiments.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(951)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:20]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute I-optimality (average prediction variance over design space)
iopt_value <- i_optimality(train_set, test_set, PC_subset)
print(paste("I-optimality value:", round(iopt_value, 6)))

# Compare I-optimality with A-optimality (both are average-based)
aopt_value <- a_optimality(train_set, test_set, PC_subset)
print(paste("A-optimality (parameter space):", round(aopt_value, 6)))
print(paste("I-optimality (design space):", round(iopt_value, 6)))

# Test different training set sizes
small_train <- sample(candidates, 12)
large_train <- sample(candidates, 28)

iopt_small <- i_optimality(small_train, test_set, PC_subset)
iopt_large <- i_optimality(large_train, test_set, PC_subset)

print(paste("Small training (n=12):", round(iopt_small, 6)))
print(paste("Large training (n=28):", round(iopt_large, 6)))
print(paste("Improvement factor:", round(iopt_small / iopt_large, 2)))
```

Description

Computes the leverage-based influence measure that was incorrectly labeled as "coefficient of determination" in the original implementation.

Usage

```
influence_measure_legacy(  
  train,  
  test = NULL,  
  P,  
  lambda = 1e-06,  
  C = NULL,  
  normalized = FALSE  
)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter (default: 1e-6)
C	Contrast matrix (default: NULL)
normalized	Whether to normalize (default: FALSE)

Details

This function computes what the original `cd_mean` actually calculated: $\text{Influence} = (\text{leverage}_i / (1 - \text{leverage}_i)^2) \times p$

This is related to Cook's distance without the residual component. For proper R^2 , use `cd_mean()` which computes average leverage.

Value

Mean influence measure value

initialize_population

Initialize random population

Description

Initialize random population

Usage

```
initialize_population(candidates, ntoselect, npop)
```

Arguments

<code>candidates</code>	Candidate individuals
<code>ntoselect</code>	Number to select per solution
<code>npop</code>	Population size

Value

List of initial solutions

legacy-compatibility *Legacy compatibility wrappers*

Description

Deprecated or compatibility wrappers retained for older STPGA code. New code should use the corresponding modern functions documented elsewhere, including `compute_amatrix()`, `population_distances()`, `distance_train_to_test_max()`, `distance_train_to_test_mean()`, `distance_internal_min()`, `distance_internal_mean()`, `distance_criterion()`, `evaluate_population()`, `subset_ga_multiobjective()`, and `subset_ga_multiobjective_single()`.

Usage

```
Amat.pieces(M, pieces = 10, mc.cores = 1)
calculate_population_distances(solutions, method = c("hamming", "jaccard", "sorensen"))
dist_to_test(train, test, Dst, lambda = NULL, C = NULL)
dist_to_test2(train, test, Dst, lambda = NULL, C = NULL)
disttoideal(X, method = "euclidean", handle_zeros = "warning")
evaluate_population_legacy(
  population,
  errorstat,
  Test,
  P,
  K = NULL,
  lambda = 1e-06,
  C = NULL,
  mc.cores = 1,
  Vg = NULL,
  Ve = NULL
)
evaluate_population_optimized(
  population,
  P,
  Test,
  errorstat = "PEVMEAN2",
  lambda = 1e-06,
  C = NULL,
```

```

    mc.cores = 1,
    K = NULL,
    Vg = NULL,
    Ve = NULL
)
GenAlgForSubsetSelectionMO(...)
GenAlgForSubsetSelectionMONoTest(...)
neg_dist_in_train(train, test = NULL, Dst, lambda = NULL, C = NULL)
neg_dist_in_train2(train, test = NULL, Dst, lambda = NULL, C = NULL)
unified_distance_criterion(
    train,
    test,
    Dst,
    criterion = c("max_to_test", "mean_to_test", "neg_min_internal", "neg_mean_internal"),
    lambda = NULL,
    C = NULL
)

```

Arguments

M, P, X	Input matrix.
pieces	Number of marker chunks for relationship matrix computation.
mc.cores	Number of parallel worker cores.
solutions, population	Candidate solutions to evaluate.
method	Distance method.
train, test, Test	Training and test set identifiers.
Dst	Distance matrix.
lambda	Ridge parameter.
C	Optional contrast matrix.
handle_zeros	How zero ranges are handled.
errorstat	Legacy criterion name.
K	Optional kinship matrix.
Vg, Ve	Genetic and residual variance components.
criterion	Distance criterion name.
...	Arguments passed to the corresponding modern implementation.

Value

The value returned by the corresponding modern implementation.

`matrix_stability_check`*Compute condition number and numerical stability metrics*

Description

Compute condition number and numerical stability metrics

Usage

```
matrix_stability_check(X)
```

Arguments

`X` Matrix to analyze

Value

List with stability metrics

`multiobjective_selection`*Multi-objective selection based on dominance ranks*

Description

Multi-objective selection based on dominance ranks

Usage

```
multiobjective_selection(fitness, ranks, n_select)
```

Arguments

`fitness` Fitness matrix
`ranks` Dominance ranks
`n_select` Number of solutions to select

Value

Indices of selected solutions

<code>mutate_solution</code>	<i>Mutate a solution</i>
------------------------------	--------------------------

Description

Mutate a solution

Usage

```
mutate_solution(solution, candidates, intensity, ntoselect)
```

Arguments

<code>solution</code>	Current solution to mutate
<code>candidates</code>	Pool of candidates
<code>intensity</code>	Mutation intensity
<code>ntoselect</code>	Target solution size

Value

Mutated solution

<code>non_dominated_sorting</code>	<i>Non-dominated sorting for multi-objective optimization</i>
------------------------------------	---

Description

Non-dominated sorting for multi-objective optimization

Usage

```
non_dominated_sorting(fitness, criteria_types)
```

Arguments

<code>fitness</code>	Fitness matrix
<code>criteria_types</code>	Vector indicating minimize (TRUE) or maximize (FALSE)

Value

Vector of dominance ranks

parallel_apply	<i>Parallel matrix operations with proper error handling</i>
----------------	--

Description

Parallel matrix operations with proper error handling

Usage

```
parallel_apply(X, FUN, mc.cores = parallel::detectCores() - 1, ...)
```

Arguments

X	List of matrices or data to process
FUN	Function to apply
mc.cores	Number of cores to use
...	Additional arguments to FUN

Value

List of results

perform_restart	<i>Perform genetic algorithm restart with population diversification</i>
-----------------	--

Description

Perform genetic algorithm restart with population diversification

Usage

```
perform_restart(  
  current_population,  
  best_solution,  
  candidates,  
  ntoselect,  
  npop,  
  generation  
)
```

Arguments

current_population	Current population
best_solution	Current best solution
candidates	Candidate pool
ntoselect	Number to select per solution
npop	Population size
generation	Current generation (for restart strategies)

Value

List with new diversified population

pev_max	<i>Maximum Prediction Error Variance (PEV)</i>
---------	--

Description

Maximum Prediction Error Variance (PEV)

Usage

```
pev_max(train, test = NULL, P, lambda = 1e-06, C = NULL, normalized = FALSE)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
normalized	Whether to normalize by trace for scale-invariance (default: FALSE)

Value

Maximum PEV value (lower is better)

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(654)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:25]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute maximum PEV (worst-case prediction uncertainty)
pev_max_value <- pev_max(train_set, test_set, PC_subset)
pev_mean_value <- pev_mean(train_set, test_set, PC_subset)

print(paste("Maximum PEV:", round(pev_max_value, 6)))
print(paste("Mean PEV:", round(pev_mean_value, 6)))
print(paste("Max/Mean ratio:", round(pev_max_value / pev_mean_value, 2)))

# Compare normalized versions
pev_max_norm <- pev_max(train_set, test_set, PC_subset, normalized = TRUE)
pev_mean_norm <- pev_mean(train_set, test_set, PC_subset, normalized = TRUE)

print(paste("Normalized Max PEV:", round(pev_max_norm, 6)))
print(paste("Normalized Mean PEV:", round(pev_mean_norm, 6)))
```

pev_mean

Mean Prediction Error Variance (PEV) - Literature Corrected

Description

Computes the mean prediction error variance for the test set given a training set. This measures the average uncertainty in predictions.

Usage

```
pev_mean(train, test = NULL, P, lambda = 1e-06, C = NULL, normalized = FALSE)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and variables as columns
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
normalized	Whether to normalize by trace for scale-invariance (default: FALSE)

Details

Corrected prediction error variance formula: $PEV = \text{Var}(y - \hat{y}) = {}^2[I + X_test(X_train'X_train)^{-1}X_test']$

The identity matrix accounts for the inherent variability of new observations. This is the correct formula according to experimental design literature.

Value

Mean PEV value (lower is better)

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for demonstration
set.seed(789)
subset_indices <- sample(1:nrow(Wheat.M), 80)
M_subset <- Wheat.M[subset_indices, 1:25]

# Extract principal components
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 15)
candidates <- setdiff(all_individuals, test_set)
train_set <- sample(candidates, 20)

# Compute mean PEV (prediction accuracy measure)
pev_value <- pev_mean(train_set, test_set, PC_subset)
print(paste("Mean PEV:", round(pev_value, 6)))

# Compare normalized vs non-normalized
pev_norm <- pev_mean(train_set, test_set, PC_subset, normalized = TRUE)
print(paste("Normalized PEV:", round(pev_norm, 6)))

# Effect of training set size on PEV
```

```
small_train <- sample(candidates, 10)
large_train <- sample(candidates, 30)

pev_small <- pev_mean(small_train, test_set, PC_subset)
pev_large <- pev_mean(large_train, test_set, PC_subset)

print(paste("Small training PEV:", round(pev_small, 6)))
print(paste("Large training PEV:", round(pev_large, 6)))
print(paste("PEV reduction factor:", round(pev_small / pev_large, 2)))
```

pev_mean_mm	<i>Mean Prediction Error Variance for Mixed Models (Henderson's BLUP)</i>
-------------	---

Description

Computes the mean prediction error variance for mixed models accounting for genetic relationships through a kinship matrix. Uses correct BLUP theory.

Usage

```
pev_mean_mm(train, test = NULL, P, K, lambda = 1e-06, C = NULL, Vg = 1, Ve = 1)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and markers as columns
K	Kinship/relationship matrix (symmetric, positive definite)
lambda	Ridge regularization parameter for numerical stability (default: 1e-6)
C	Contrast matrix for specific linear combinations (default: NULL)
Vg	Genetic variance component (default: 1)
Ve	Error variance component (default: 1)

Details

Implements the correct BLUP prediction error variance formula from Henderson (1984):

$$PEV = \sigma_u^2 [G_{22} - G_{21} V_{11}^{-1} G_{12} - G_{21} V_{11}^{-1} X_1 (X_1' V_{11}^{-1} X_1)^{-1} X_1' V_{11}^{-1} G_{12}]$$
 where $V_{11} = \sigma_u^2 G_{11} + \sigma_e^2 I$ and both correction terms are SUBTRACTED.

Value

Mean PEV value for mixed models (lower is better)

References

Henderson, C.R. (1984). Applications of Linear Models in Animal Breeding. Searle, S.R., Casella, G., McCulloch, C.E. (1992). Variance Components.

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a subset for mixed model demonstration
set.seed(987)
subset_indices <- sample(1:nrow(Wheat.M), 60)
M_subset <- Wheat.M[subset_indices, 1:50]
K_subset <- Wheat.K[subset_indices, subset_indices]

# Define train and test sets
all_individuals <- rownames(M_subset)
test_set <- sample(all_individuals, 12)
train_set <- sample(setdiff(all_individuals, test_set), 18)

# Mixed model PEV with different heritability scenarios
h2_values <- c(0.2, 0.5, 0.8)

print("PEV Mean MM - Different Heritabilities:")
for (h2 in h2_values) {
  var_comps <- h2_to_variances(h2)
  pev_mm <- pev_mean_mm(train_set, test_set, M_subset, K_subset,
                        Vg = var_comps$Vg, Ve = var_comps$Ve)
  print(paste("h2 =", h2, "-> PEV =", round(pev_mm, 6)))
}

# Compare with non-mixed model PEV using PCA
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)
pev_regular <- pev_mean(train_set, test_set, PC_subset)

print(paste("Regular PEV (PCA):", round(pev_regular, 6)))
print("Mixed model accounts for genetic relationships via kinship matrix")
```

pev_mean_mm_h2

Mean Prediction Error Variance for Mixed Models using heritability

Description

Mean Prediction Error Variance for Mixed Models using heritability

Usage

```
pev_mean_mm_h2(
  train,
  test = NULL,
  P,
  K,
  h2,
  lambda = 1e-06,
  C = NULL,
  total_var = 1,
  normalized = FALSE
)
```

Arguments

train	Vector of training set individual names
test	Vector of test set individual names (if NULL, uses training set)
P	Prediction matrix with individuals as rows and markers as columns
K	Kinship/relationship matrix (symmetric, positive definite)
h2	Heritability (between 0 and 1)
lambda	Ridge regularization parameter (default: 1e-6)
C	Contrast matrix (default: NULL)
total_var	Total variance (default: 1)
normalized	If TRUE, return PEV as proportion of total variance

Value

Mean PEV value

`plot_pareto_progress` *Plot Pareto front progress*

Description

Plot Pareto front progress

Usage

```
plot_pareto_progress(fitness, criteria, directions, generation, axes_labels)
```

Arguments

fitness	Fitness matrix
criteria	Criteria names
directions	Plot directions
generation	Current generation
axes_labels	Axis labels

population_distances	<i>Calculate population diversity metrics</i>
----------------------	---

Description

Calculate population diversity metrics

Usage

```
population_distances(solutions, method = c("hamming", "jaccard", "sorensen"))
```

Arguments

solutions	List of solutions in the population
method	Distance calculation method: "hamming", "jaccard", "sorensen" (default: "jaccard")

Value

Distance matrix or diversity summary statistics

rank_selection	<i>Rank-based selection with selection pressure control</i>
----------------	---

Description

Rank-based selection with selection pressure control

Usage

```
rank_selection(  
  population,  
  fitness,  
  n_select,  
  selection_pressure = 1.5,  
  method = "linear"  
)
```

Arguments

population	List of solutions
fitness	Vector of fitness values
n_select	Number of individuals to select
selection_pressure	Selection pressure parameter (1.0-2.0, default: 1.5)
method	Rank selection method: "linear", "exponential" (default: "linear")

Value

Indices of selected individuals

Examples

```
# Create a sample population and fitness values
set.seed(123)
population <- list(
  c("ind1", "ind2", "ind3"),
  c("ind2", "ind4", "ind5"),
  c("ind1", "ind3", "ind6"),
  c("ind4", "ind5", "ind6"),
  c("ind1", "ind4", "ind7")
)

# Fitness values (lower is better for minimization)
fitness <- c(10.5, 8.2, 12.1, 9.8, 7.5)

# Rank-based selection with moderate pressure
selected_indices <- rank_selection(population, fitness, n_select = 3,
                                  selection_pressure = 1.4, method = "linear")

print("Selected indices:")
print(selected_indices)
print("Corresponding fitness values:")
print(fitness[selected_indices])

# Compare different selection pressures
low_pressure <- rank_selection(population, fitness, 3, 1.1, "linear")
high_pressure <- rank_selection(population, fitness, 3, 1.8, "linear")

print("Low pressure selection:")
print(fitness[low_pressure])
print("High pressure selection:")
print(fitness[high_pressure])

# Exponential ranking
exp_selection <- rank_selection(population, fitness, 3, 1.3, "exponential")
print("Exponential ranking selection:")
print(fitness[exp_selection])
```

ridge_regression_cv	<i>Ridge regression with optimal lambda selection</i>
---------------------	---

Description

Ridge regression with optimal lambda selection

Usage

```
ridge_regression_cv(y, X, lambda_seq = NULL, cv_folds = 5, criterion = "mse")
```

Arguments

y	Response vector
X	Design matrix
lambda_seq	Sequence of lambda values to test (default: NULL for automatic)
cv_folds	Number of cross-validation folds (default: 5)
criterion	Selection criterion: "mse", "aic", "bic" (default: "mse")

Value

List with optimal lambda and fitted model

safe_matrix_inverse	<i>Efficient matrix inversion with numerical stability</i>
---------------------	--

Description

Efficient matrix inversion with numerical stability

Usage

```
safe_matrix_inverse(X, lambda = 0, method = "cholesky")
```

Arguments

X	Matrix to invert
lambda	Ridge parameter for regularization (default: 0)
method	Inversion method: "cholesky", "svd", "eigen" (default: "cholesky")

Value

Inverse matrix

`select_diverse_subset`*Select diverse subset using crowding distance*

Description

Select diverse subset using crowding distance

Usage

```
select_diverse_subset(fitness, max_size)
```

Arguments

<code>fitness</code>	Fitness matrix
<code>max_size</code>	Maximum number of solutions to select

Value

Indices of selected solutions

`simple_cache`*Simple hash-based cache for computation results*

Description

Simple hash-based cache for computation results

Usage

```
simple_cache(key, value = NULL, cache_env = NULL)
```

Arguments

<code>key</code>	Cache key (will be hashed)
<code>value</code>	Value to cache (only if not retrieving)
<code>cache_env</code>	Environment to store cache (default: private environment)

Value

Cached value if exists, NULL otherwise

<code>standardize_name</code>	<i>Naming Standards and Conventions</i>
-------------------------------	---

Description

Functions to ensure consistent naming throughout the package

Usage

```
standardize_name(name)
```

Arguments

<code>name</code>	Function or variable name
-------------------	---------------------------

Value

Standardized snake_case name

Author(s)

Deniz Akdemir Convert function names to snake_case

<code>standardize_parameters</code>	<i>Standardize common parameter names across functions</i>
-------------------------------------	--

Description

Standardize common parameter names across functions

Usage

```
standardize_parameters(...)
```

Arguments

<code>...</code>	Named parameters to standardize
------------------	---------------------------------

Value

List with standardized parameter names and values

subset_ga

STPGA Core Genetic Algorithm Functions

Description

Clean, well-organized genetic algorithm functions for subset selection

Usage

```
subset_ga(  
  P,  
  Candidates,  
  Test = NULL,  
  ntoselect,  
  npop = 100,  
  nelite = 5,  
  keepbest = TRUE,  
  tabu = TRUE,  
  tabumemsize = 1,  
  mutprob = 0.8,  
  mutintensity = 1,  
  niterations = 500,  
  minitbefstop = 200,  
  niterreg = 5,  
  convergence_window_multiplier = 4,  
  enable_restart = TRUE,  
  restart_threshold = 0.5,  
  max_restarts = 2,  
  lambda = 1e-06,  
  plotiters = FALSE,  
  plotype = 1,  
  criterion = "PEVMEAN2",  
  C = NULL,  
  mc.cores = 1,  
  InitPop = NULL,  
  tolconv = 1e-07,  
  Vg = NULL,  
  Ve = NULL,  
  Fedorov = FALSE,  
  adaptive_mutation = TRUE,  
  selection_method = "tournament",  
  tournament_size = 3,  
  selection_pressure = 1.5,  
  diversity_preservation = TRUE,  
  diversity_method = "crowding",  
  crowding_factor = 3,  
  sharing_radius = 0.1,  
)
```

```

    crossover_method = "adaptive",
    diversity_target = 0.3,
    verbose = FALSE
)

```

Arguments

P	Prediction matrix (individuals x markers/predictors)
Candidates	Vector of candidate individual names
Test	Vector of test individual names
ntoselect	Number of individuals to select for training
npop	Population size (default: 100)
nelite	Number of elite individuals to keep (default: 5)
keepbest	Whether to keep best solution across generations (default: TRUE)
tabu	Whether to use tabu search (default: TRUE)
tabumemsize	Tabu memory size (default: 1)
mutprob	Mutation probability (default: 0.8)
mutintensity	Mutation intensity (default: 1)
niterations	Maximum iterations (default: 500)
minitbefstop	Minimum iterations before stopping (default: 200)
niterreg	Iterations for convergence check (default: 5)
convergence_window_multiplier	Multiplier for convergence window size (default: 4)
enable_restart	Whether to enable restart mechanism for premature convergence (default: TRUE)
restart_threshold	Threshold for triggering restart (default: 0.5)
max_restarts	Maximum number of restarts allowed (default: 2)
lambda	Ridge regression parameter (default: 1e-6)
plotiters	Whether to plot during iterations (default: FALSE)
plottype	Type of plot (default: 1)
criterion	Optimization criterion (default: "PEVMEAN2")
C	Contrast matrix (default: NULL)
mc.cores	Number of cores for parallel processing (default: 1)
InitPop	Initial population (default: NULL)
tolconv	Convergence tolerance (default: 1e-7)
Vg	Genetic variance matrix for mixed models (default: NULL)
Ve	Error variance matrix for mixed models (default: NULL)
Fedorov	Whether to use Fedorov exchange (default: FALSE)

adaptive_mutation	Whether to use adaptive mutation rates (default: TRUE)
selection_method	Selection method: "tournament", "elite", "rank", "hybrid" (default: "tournament")
tournament_size	Tournament size for tournament selection (default: 3)
selection_pressure	Selection pressure for rank-based selection (default: 1.5)
diversity_preservation	Whether to preserve diversity (default: TRUE)
diversity_method	Diversity method: "crowding", "sharing", "both" (default: "crowding")
crowding_factor	Factor for crowding replacement (default: 3)
sharing_radius	Radius for fitness sharing (default: 0.1)
crossover_method	Crossover method (default: "adaptive")
diversity_target	Target diversity level (default: 0.3)
verbose	Whether to print verbose output (default: FALSE)

Value

List containing best solution, fitness history, and statistics

Author(s)

Deniz Akdemir Primary genetic algorithm for subset selection with training/test split

Examples

```
# Load wheat genomic data
data(WheatData)

# Create a manageable subset for demonstration
set.seed(123)
subset_indices <- sample(1:nrow(Wheat.M), 100)
M_subset <- Wheat.M[subset_indices, 1:30]

# Extract principal components for genetic algorithm
pca_result <- prcomp(M_subset, center = TRUE, scale. = TRUE)
PC_subset <- pca_result$x[, 1:5]
rownames(PC_subset) <- rownames(M_subset)

# Define candidate and test sets
all_individuals <- rownames(PC_subset)
test_set <- sample(all_individuals, 20)
```

```

candidates <- setdiff(all_individuals, test_set)

# Basic genetic algorithm run
ga_result <- subset_ga(
  P = PC_subset,
  Candidates = candidates,
  Test = test_set,
  ntoselect = 15,
  npop = 20,
  niterations = 30,
  criterion = "pev_mean",
  verbose = FALSE
)

print(paste("Best fitness:", round(ga_result$best_fitness, 6)))
print(paste("Convergence generation:", ga_result$convergence_generation))
print(paste("Selected individuals:", length(ga_result$best_solution)))

# Enhanced genetic algorithm with rank-based selection and restart
ga_enhanced <- subset_ga(
  P = PC_subset,
  Candidates = candidates,
  Test = test_set,
  ntoselect = 15,
  npop = 20,
  niterations = 30,
  criterion = "pev_mean",
  selection_method = "rank",
  selection_pressure = 1.4,
  enable_restart = TRUE,
  restart_threshold = 0.6,
  max_restarts = 1,
  verbose = FALSE
)

print(paste("Enhanced GA fitness:", round(ga_enhanced$best_fitness, 6)))
print(paste("Restart count:", ga_enhanced$restart_count))

# Generate convergence diagnostics
diags <- convergence_diagnostics(ga_enhanced, plot = FALSE)
print(paste("Converged:", diags$convergence_status$converged))
print(paste("Convergence reason:", diags$convergence_status$reason))

```

subset_ga_multiobjective

STPGA Multi-Objective Genetic Algorithm

Description

Multi-objective optimization for subset selection

Usage

```
subset_ga_multiobjective(
  Pcs = NULL,
  Dist = NULL,
  Kernel = NULL,
  candidates,
  test,
  ntoselect,
  criteria,
  criteria_types,
  plot_directions,
  npop = 100,
  mutprob = 0.8,
  mutintensity = 1,
  niterations = 500,
  lambda = 1e-06,
  plot_iterations = FALSE,
  mc.cores = 1,
  initial_population = NULL,
  C = NULL,
  axes_labels = NULL,
  archive_size = 200,
  diversity_maintenance = TRUE
)
```

Arguments

Pcs	Principal components matrix (can be NULL)
Dist	Distance matrix (can be NULL)
Kernel	Kernel matrix (can be NULL)
candidates	Vector of candidate individual names
test	Vector of test individual names
ntoselect	Number of individuals to select for training
criteria	Vector of optimization criteria names
criteria_types	Vector indicating minimize/maximize for each criterion
plot_directions	Vector for plotting directions (1 or -1 for each criterion)
npop	Population size (default: 100)
mutprob	Mutation probability (default: 0.8)
mutintensity	Mutation intensity (default: 1)
niterations	Maximum iterations (default: 500)
lambda	Ridge regression parameter (default: 1e-6)
plot_iterations	Whether to plot during iterations (default: FALSE)

<code>mc.cores</code>	Number of cores for parallel processing (default: 1)
<code>initial_population</code>	Initial population (default: NULL)
<code>C</code>	Contrast matrix (default: NULL)
<code>axes_labels</code>	Labels for plot axes (default: NULL)
<code>archive_size</code>	Maximum size of Pareto archive (default: 200)
<code>diversity_maintenance</code>	Whether to maintain diversity (default: TRUE)

Value

List containing Pareto front and optimization results

Author(s)

Deniz Akdemir Multi-objective genetic algorithm for subset selection with test set

`subset_ga_multiobjective_single`
Multi-objective GA without test set (single population)

Description

Multi-objective GA without test set (single population)

Usage

```
subset_ga_multiobjective_single(
  Pcs = NULL,
  Dist = NULL,
  Kernel = NULL,
  candidates,
  ntoselect,
  criteria,
  criteria_types,
  plot_directions,
  ...
)
```

Arguments

<code>Pcs</code>	Principal components matrix
<code>Dist</code>	Distance matrix
<code>Kernel</code>	Kernel matrix
<code>candidates</code>	Candidate individuals

<code>ntoselect</code>	Number to select
<code>criteria</code>	Optimization criteria
<code>criteria_types</code>	Minimize/maximize indicators
<code>plot_directions</code>	Plot directions
<code>...</code>	Additional parameters

Value

Multi-objective optimization results

<code>subset_ga_single</code>	<i>Genetic algorithm for subset selection without test set (single objective)</i>
-------------------------------	---

Description

Genetic algorithm for subset selection without test set (single objective)

Usage

```
subset_ga_single(P, ntoselect, ...)
```

Arguments

<code>P</code>	Prediction matrix (individuals x markers/predictors)
<code>ntoselect</code>	Number of individuals to select
<code>...</code>	Additional parameters (same as subset_ga but without Test parameter)

Value

List containing best solution and statistics

tournament_selection	<i>Tournament selection</i>
----------------------	-----------------------------

Description

Tournament selection

Usage

```
tournament_selection(population, fitness, tournament_size = 3, n_select = 1)
```

Arguments

population	List of solutions
fitness	Vector of fitness values
tournament_size	Size of tournament (default: 3)
n_select	Number of individuals to select (default: 1)

Value

Indices of selected individuals

transform_fitness	<i>Fitness transformation and normalization utilities</i>
-------------------	---

Description

Fitness transformation and normalization utilities

Usage

```
transform_fitness(fitness, method = "linear", scaling_factor = 2)
```

Arguments

fitness	Vector of fitness values
method	Transformation method: "linear", "rank", "tournament" (default: "linear")
scaling_factor	Scaling factor for transformations (default: 2)

Value

Transformed fitness values

update_pareto_archive	<i>Update Pareto archive with new solutions</i>
-----------------------	---

Description

Update Pareto archive with new solutions

Usage

```
update_pareto_archive(  
    population,  
    fitness,  
    archive,  
    archive_fitness,  
    max_size,  
    criteria_types  
)
```

Arguments

- population Current population
- fitness Fitness matrix
- archive Current archive
- archive_fitness Archive fitness matrix
- max_size Maximum archive size
- criteria_types Optimization types (minimize/maximize)

Value

Updated archive and fitness

update_progress	<i>Update progress bar</i>
-----------------	----------------------------

Description

Update progress bar

Usage

```
update_progress(pb, increment = 1)
```

Arguments

pb	Progress bar object
increment	How much to increment (default: 1)

```
validate_kinship_matrix
```

Validate kinship matrix

Description

Validate kinship matrix

Usage

```
validate_kinship_matrix(K, individual_names)
```

Arguments

K	Kinship matrix
individual_names	Names of individuals to check against

Value

NULL if valid, throws error if invalid

```
validate_matrix_params
```

Input Validation and Utility Functions

Description

Comprehensive validation functions for STPGA package

Usage

```
validate_matrix_params(
  P,
  K = NULL,
  lambda = 1e-06,
  train = NULL,
  test = NULL,
  C = NULL
)
```

Arguments

P	Prediction matrix (individuals x variables)
K	Kinship matrix (optional, for mixed models)
lambda	Ridge regularization parameter
train	Training set names
test	Test set names (optional)
C	Contrast matrix (optional)

Value

NULL if valid, throws error if invalid

Author(s)

Deniz Akdemir Validate matrix parameters for optimization criteria

`validate_selection_parameters`

Validate and adjust selection parameters

Description

Validate and adjust selection parameters

Usage

```
validate_selection_parameters(
  selection_method,
  tournament_size = 3,
  selection_pressure = 1.5,
  population_size
)
```

Arguments

selection_method	Selection method name
tournament_size	Tournament size (for tournament selection)
selection_pressure	Selection pressure (for rank selection)
population_size	Population size

Value

List with validated parameters and warnings

Examples

```
# Test parameter validation with valid parameters
validation_good <- validate_selection_parameters(
  selection_method = "rank",
  tournament_size = 3,
  selection_pressure = 1.5,
  population_size = 50
)

print("Valid parameters:")
print(paste("Method:", validation_good$selection_method))
print(paste("Tournament size:", validation_good$tournament_size))
print(paste("Selection pressure:", validation_good$selection_pressure))
print(paste("Warnings:", length(validation_good$warnings)))

# Test parameter validation with invalid parameters
validation_bad <- validate_selection_parameters(
  selection_method = "invalid_method",
  tournament_size = 100, # Too large
  selection_pressure = 3.0, # Out of range
  population_size = 25
)

print("Invalid parameters corrected:")
print(paste("Method corrected to:", validation_bad$selection_method))
print(paste("Tournament size corrected to:", validation_bad$tournament_size))
print(paste("Selection pressure corrected to:", validation_bad$selection_pressure))
print(paste("Number of warnings:", length(validation_bad$warnings)))

for (warning in validation_bad$warnings) {
  print(paste("Warning:", warning))
}
```

```
validate_variance_components
```

Validate variance components

Description

Validate variance components

Usage

```
validate_variance_components(Vg, Ve, h2 = NULL)
```

Arguments

Vg	Genetic variance component
Ve	Error variance component
h2	Heritability (optional, for cross-validation)

Value

NULL if valid, throws error if invalid

WheatData	<i>Adult plant height (estimated genetic values) for 1182 elite wheat lines</i>
------------------	---

Description

Containing the phenotypic observations 'Wheat.Y', markers 'Wheat.M' and genetic relationships 'Wheat.K'. The 4670 markers available for these 200 genotypes were pre-processed for missingness and minor allele frequencies, coded numerically as -1, 0, and 1; the relationship genomic relationship matrix was calculated from these markers.

Source

This dataset was obtained from <https://triticeaetoolbox.org/>.

Index

a_optimality, 4
adaptive_mutation_rate, 5
Amat.pieces (*legacy-compatibility*), 51
block_matrix_mult, 6
cache_stats, 7
calculate_crowding_distance, 7
calculate_diversity, 8
calculate_population_distances (*legacy-compatibility*), 51
cd_mean, 8
cd_mean_mm, 9
cd_mean_mm_h2, 11
check_convergence, 11
check_dominance, 12
check_matrix_dimensions, 13
check_memory_usage, 13
check_multi_criteria_convergence, 14
check_parameter_naming, 14
chunked_crossprod, 15
chunked_matrix_operation, 15
clear_cache, 16
compute_amatrix, 16
compute_convergence_metrics, 17
compute_generation_stats, 17
compute_population_stats, 18
compute_prediction_core, 18
convergence_diagnostics, 19
create_evaluation_cache, 20
create_function_alias, 21
create_progress_bar, 21
criterion, 22
crossover, 23
crowding_replacement, 24
d_optimality, 24
determine_optimal_batch_size, 26
determine_primary_matrix, 26
dist_to_test (*legacy-compatibility*), 51
dist_to_test2 (*legacy-compatibility*), 51
distance_criterion, 27
distance_internal_mean, 28
distance_internal_min, 28
distance_to_ideal, 29
distance_train_to_test_max, 30
distance_train_to_test_mean, 30
disttoideal (*legacy-compatibility*), 51
diversity_preserving_selection, 31
diversity_summary, 32
document_parameter, 32
e_optimality, 33
elite_selection, 34
environmental_selection, 35
evaluate_multiobjective_population, 35
evaluate_population, 36
evaluate_population_legacy (*legacy-compatibility*), 51
evaluate_population_optimized (*legacy-compatibility*), 51
evaluate_population_smart, 37
evaluate_solution_batch, 38
extract_pareto_front, 39
find_non_dominated, 39
finish_progress, 40
fitness_sharing, 40
g_optimality, 41
GenAlgForSubsetSelectionMO (*legacy-compatibility*), 51
GenAlgForSubsetSelectionMONoTest (*legacy-compatibility*), 51
generate_doc_template, 42

generate_multiobjective_offspring, 43
generate_offspring, 43
genomic_relationship_matrix, 44
get_adaptive_ridge, 45
get_function_variants, 46
get_optimal_cores, 46
group_similar_solutions, 47

h2_to_variances, 47

i_optimality, 48
influence_measure_legacy, 49
initialize_population, 50

legacy-compatibility, 51

matrix_stability_check, 53
multiobjective_selection, 53
mutate_solution, 54

neg_dist_in_train
 (*legacy-compatibility*), 51
neg_dist_in_train2
 (*legacy-compatibility*), 51
non_dominated_sorting, 54

parallel_apply, 55
perform_restart, 55
pev_max, 56
pev_mean, 57
pev_mean_mm, 59
pev_mean_mm_h2, 60
plot_pareto_progress, 61
population_distances, 62

rank_selection, 62
ridge_regression_cv, 64

safe_matrix_inverse, 64
select_diverse_subset, 65
simple_cache, 65
standardize_name, 66
standardize_parameters, 66
subset_ga, 67
subset_ga_multiobjective, 70
subset_ga_multiobjective_single, 72
subset_ga_single, 73

tournament_selection, 74
transform_fitness, 74

unified_distance_criterion
 (*legacy-compatibility*), 51
update_pareto_archive, 75
update_progress, 75

validate_kinship_matrix, 76
validate_matrix_params, 76
validate_selection_parameters, 77
validate_variance_components, 78

Wheat.K (*WheatData*), 79
Wheat.M (*WheatData*), 79
Wheat.Y (*WheatData*), 79
WheatData, 79