

Package: synergyMixR (via r-universe)

October 11, 2026

Type Package

Title Mixture-Based Muscle Synergy Analysis

Version 1.0.0

Author Yusuke Matsui [aut, cre]

Maintainer Yusuke Matsui <matsui@nagoya-u.jp>

Description Provides tools for analyzing muscle synergy using various statistical methods including MPCA and MFA. Enables researchers to identify distinct subgroups within populations that exhibit different muscle synergy patterns, rather than forcing all subjects into a single global model.

License GPL-3 + file LICENSE

URL <https://github.com/matsui-lab/synergyMixR>

BugReports <https://github.com/matsui-lab/synergyMixR/issues>

Encoding UTF-8

SystemRequirements GNU make

LazyData true

Depends R (>= 4.1.0)

RoxygenNote 7.3.3

Imports stats, utils, Rcpp, mvtnorm, parallel, magrittr, signal, tidyselect, ggplot2

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Suggests dplyr, tidyr, reshape2, mclust, ggrepel, stringr, clue, rlang, knitr, rmarkdown, psych, viridis, patchwork, uwot, pbapply, testthat, future, future.apply, future.batchtools, progressr, purrr, umap

Config/pak/sysreqs make

Repository <https://matsui-lab.r-universe.dev>

Date/Publication 2026-02-21 05:28:46 UTC

RemoteUrl <https://github.com/matsui-lab/synergyMixR>

RemoteRef HEAD

RemoteSha 7178667190d3f0f52d7e571056970ef07b6d20f0

Contents

aggregate_partial_results	5
align_clusters_to_truth	6
align_factor_signs_mfa	7
align_factor_signs_mPCA	8
assign_by_kmeans	9
bandpass_filter	10
bootstrap_loading_ci	11
calc_reconstruction_error_mixtureFA	13
calc_reconstruction_error_mixturePCA	14
calc_reconstruction_error_singleFA	14
calc_reconstruction_error_singlePCA	15
calc_total_SST	15
calibrate_model	16
check_future_available	16
check_pbs_job_status	17
check_SSE_diagnostics	17
compare_baselines	20
compare_static_vs_dynamic	23
compute_ari	26
compute_ari_vaf_table	27
compute_BIC	27
compute_BIC_mfa	28
compute_BIC_mPCA	29
compute_bic_wrapper	30
compute_calibration	31
compute_cluster_sizes	32
compute_cluster_sizes_mPCA	33
compute_clusterwise_subspace_score	33
compute_ece	35
compute_entropy	36
compute_factor_scores	37
compute_factor_scores_mfa	38
compute_factor_scores_mPCA	39
compute_factor_scores_mPCA_single	40
compute_factor_scores_optimized	40
compute_global_vaf_mfa	41
compute_global_vaf_mPCA	42
compute_ICL	43
compute_ICL_mfa	44
compute_ICL_mPCA	45
compute_logLik	46

compute_logLik_mfa	47
compute_logLik_mPCA	47
compute_mvc_reference	48
compute_reconstruction_error	49
compute_smoothness_penalty	50
compute_subspace_similarity	51
compute_VAF	52
compute_VAF_mfa	53
compute_VAF_mPCA	54
compute_vaf_wrapper	54
compute_vss_criterion	55
count_completed_tasks	57
emg_preset_config	57
evaluate_model_selection	58
extract_cluster_labels	59
extract_envelope	60
extract_selection_summary	62
extract_subject_features_by_singlePCA	62
facet_mixsynergy	63
fit_single_factor_analysis	64
fit_single_pca	65
fit_two_step_baseline	65
generate_pbs_script	67
generate_synergy_data	68
get_model_by_K_r	71
get_model_by_K_r_mPCA	72
make_table1	72
make_table2	73
mfa_em_fit	73
mfa_em_fit_cpp_singleInit	76
mixture_pca_em_fit	77
mixture_pca_em_fit_cpp_singleInit	80
normalize_emg	81
notch_filter	83
plot.mixture_r_selection	84
plot_activation_patterns	85
plot_aligned_basis_heatmaps	86
plot_all_factor_scores_mfa	87
plot_all_factor_scores_mPCA	88
plot_all_reconstructed_waveforms_mfa	89
plot_all_reconstructed_waveforms_mPCA	90
plot_basis_umap	91
plot_bic_comparison	93
plot_bic_line	94
plot_calibration	94
plot_cluster_embedding	96
plot_cluster_synergy_loadings_mfa	97
plot_cluster_synergy_loadings_mPCA	98

plot_confusion_matrix	99
plot_fig1_ARI	100
plot_fig1_baseline_comparison	101
plot_fig2_BIC_heatmap	102
plot_fig3_init_boxplot	103
plot_fig4to6_reconstruction_vaf	103
plot_metric_vs_axis	104
plot_neural_command	105
plot_noise_profiles	106
plot_reconstructed_emg	107
plot_sse_heatmap	108
plot_sweep_summary	109
plot_synergy_bases	110
plot_synergy_correlation_comparison	111
plot_synergy_summary	113
plot_waveform_comparison_mfa	114
plot_waveform_comparison_mPCA	114
plot_winrate_heatmap	115
posthoc_mfa_evaluation	116
posthoc_mPCA_evaluation	117
preprocess_emg	118
print.bootstrap_ci	120
print.calibration_result	121
print.mixture_r_selection	121
print.preprocessed_emg	122
read_all_results	122
read_and_visualize_all	123
reconstruct_singleMPCA	124
rectify	124
remove_dc_offset	125
rotate_mfa_model	126
rotate_mPCA_model	127
run_k_selection_comparison	128
run_param_sweep	129
run_param_sweep_baselines	131
run_param_sweep_modelsel	135
run_single_task	139
run_sweep_future	141
scale_color_mixsynergy	142
scale_fill_mixsynergy	143
scale_fill_mixsynergy_discrete	143
scale_fill_mixsynergy_diverging	144
select_K_bootstrap_stability	145
select_K_crossvalidation	146
select_K_r	147
select_optimal_K_r_mfa	150
select_optimal_K_r_mfa_twostage	151
select_optimal_K_r_mPCA	153

select_optimal_K_r_mPCA_twostage	155
select_r_by_smoothness_mfa	157
select_r_by_smoothness_mPCA	158
select_r_by_vaf_mfa	160
select_r_by_vaf_mPCA	161
select_r_by_vss_mfa	162
select_r_by_vss_mPCA	164
select_r_clusterwise_mfa	166
select_r_clusterwise_mPCA	170
select_r_global_mfa	173
select_r_global_mPCA	175
select_r_mixture	178
select_r_multicriteria_mfa	181
select_r_multicriteria_mPCA	183
setup_future_backend	185
show_parallel_options	187
simulate_dynamic_synergy_data	187
simulation_comparison	190
simulation_visualization	190
strip_attributes	191
submit_hpc_sweep	191
submit_pbs_sweep	193
summarize_vaf_sse_mfa	194
summary.mixture_r_selection	195
summary.preprocessed_emg	195
theme_mixsynergy	196
theme_mixsynergy_heatmap	196
theme_sweep	197
visualize_twostep_diagnostics	198
wait_for_pbs_job	198

Index**200**

 aggregate_partial_results

Aggregate Partial Results from PBS Jobs

Description

Collects and combines results from individual PBS array job tasks.

Usage

```

aggregate_partial_results(
  output_dir,
  pattern = "result_.*\\.rds$",
  remove_partial = FALSE,
  verbose = TRUE
)
```

Arguments

output_dir	Character. Directory containing partial results.
pattern	Character. Regex pattern for result files.
remove_partial	Logical. If TRUE, remove partial result files after aggregation.
verbose	Logical. Print progress messages.

Value

Data frame with combined results, or NULL if no results found.

Examples

```
## Not run:
# After PBS jobs complete
results <- aggregate_partial_results("results_sweep_modelsel")

# View results
head(results)

## End(Not run)
```

align_clusters_to_truth

Align Cluster Labels to Ground Truth Using Hungarian Algorithm

Description

Finds the optimal mapping from estimated cluster labels to true cluster labels using the Hungarian algorithm. This resolves the label switching problem inherent in clustering.

Usage

```
align_clusters_to_truth(z_true, z_hat)
```

Arguments

z_true	Integer vector of true cluster assignments (1 to K_true).
z_hat	Integer vector of estimated cluster assignments (1 to K_hat).

Details

The Hungarian algorithm (via `clue::solve_LSAP`) finds the assignment that maximizes the overlap between estimated and true clusters. This is essential for fair evaluation of clustering methods where label identities are arbitrary.

When $K_{\text{hat}} \neq K_{\text{true}}$, the algorithm handles the mismatch by:

- If $K_{\text{hat}} < K_{\text{true}}$: Some true clusters may not have a match
- If $K_{\text{hat}} > K_{\text{true}}$: Some estimated clusters map to the same true cluster

Value

A list with:

mapping Named integer vector where names are estimated labels and values are corresponding true labels.

z_aligned Integer vector of aligned estimated labels (same length as z_hat, with labels remapped to match z_true).

accuracy Numeric scalar: proportion of correctly matched assignments after alignment.

Returns a list with all NA values if inputs are invalid.

Examples

```
## Not run:
z_true <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
z_hat <- c(3, 3, 3, 1, 1, 1, 2, 2, 2) # Labels are permuted
result <- align_clusters_to_truth(z_true, z_hat)
result$accuracy # Should be 1.0 (perfect after alignment)

## End(Not run)
```

align_factor_signs_mfa

Align (Flip) Factor Signs in an MFA Model

Description

After running [rotate_mfa_model](#) or any rotation, each factor's sign can be arbitrary. This function enforces a rule such as: "The variable with the largest absolute loading in each factor must be positive." Then, if that factor's column is flipped, we also flip the corresponding column in the factor scores, to keep the model consistent.

Usage

```
align_factor_signs_mfa(mfa_fit, method = c("max_abs"), cluster_ids = NULL)
```

Arguments

mfa_fit	A fitted MFA model (with Lambda[[k]], factor_scores[[i]], z, etc.).
method	A string specifying how to choose the sign. Currently only "max_abs" is implemented, which flips the column if the variable with the largest absolute loading is negative.
cluster_ids	Which clusters to process. Defaults to all.

Details

- For each cluster k , we loop over each factor (column). We find the row index $i_{\max}$ that has the largest absolute loading. If $\Lambda[k][i_{\max}, \text{factor}] < 0$, we multiply that entire column by -1 . - If `factor_scores` is present, we also multiply the corresponding column in each subject's factor score by -1 to maintain consistency. - This doesn't change the model fit or BIC, just the sign convention.

Value

The same `mfa_fit` object, but with possibly flipped signs in `Lambda` columns and `factor_scores` columns.

`align_factor_signs_mPCA`

Align (Flip) Factor Signs in a Mixture PCA Model

Description

Similar to [align_factor_signs_mfa](#), but for a Mixture PCA model whose loadings are in $W[[k]]$.

Usage

```
align_factor_signs_mPCA(mPCA_fit, method = c("max_abs"), cluster_ids = NULL)
```

Arguments

<code>mPCA_fit</code>	A fitted MPCA model (with $W[[k]]$, <code>factor_scores[[i]]</code> , <code>z</code> , etc.).
<code>method</code>	A string specifying how to choose the sign. Defaults to "max_abs".
<code>cluster_ids</code>	Which clusters to process. Defaults to all.

Details

- For each cluster k , we loop over each factor (column) in $W[[k]]$. If the entry with largest absolute loading is negative, we multiply that entire column by -1 . - If `factor_scores` is present, we also multiply the corresponding column in each subject's factor scores by -1 .

Value

The same `mPCA_fit` object, but with possibly flipped signs in `W` columns and `factor_scores` columns (if present).

assign_by_kmeans	<i>K-means-based cluster assignment</i>
------------------	---

Description

Performs k-means clustering on subject-level features to obtain initial cluster assignments for mixture model fitting.

Usage

```
assign_by_kmeans(features, K)
```

Arguments

features	An (N x d) matrix of subject-level features, where N is the number of subjects and d is the feature dimension.
K	Number of clusters to create.

Details

This function uses the standard k-means algorithm with 10 random starts (nstart=10) to find a stable clustering solution. The resulting cluster assignments can be used as initial values for EM algorithms in mixture models.

Value

An integer vector of length N containing cluster labels (1..K). Each element indicates which cluster the corresponding subject is assigned to.

Examples

```
## Not run:  
# Create example features  
features <- matrix(rnorm(100), nrow = 10, ncol = 10)  
  
# Assign to 3 clusters  
z_init <- assign_by_kmeans(features, K = 3)  
table(z_init) # Check cluster sizes  
  
## End(Not run)
```

bandpass_filter	<i>Bandpass Filter for EMG Signals</i>
-----------------	--

Description

Applies a bandpass filter to EMG signals to extract the frequency band of interest (typically 20-450 Hz for surface EMG).

Usage

```
bandpass_filter(
    data,
    sampling_rate,
    low_cutoff = 20,
    high_cutoff = 450,
    filter_order = 4,
    filter_type = "butterworth",
    zero_phase = TRUE
)
```

Arguments

data	EMG signal. Can be a numeric vector (single channel), a matrix (T x M, time x channels), or a list of matrices.
sampling_rate	Sampling frequency in Hz.
low_cutoff	Lower cutoff frequency in Hz. Default is 20 Hz.
high_cutoff	Upper cutoff frequency in Hz. Default is 450 Hz.
filter_order	Filter order. Default is 4 (creates 4th order filter, 8th order effective with zero-phase filtering).
filter_type	Type of filter: "butterworth" (default), "chebyshev1", or "chebyshev2".
zero_phase	Logical. If TRUE (default), applies zero-phase filtering (forward-backward filtering) to avoid phase distortion.

Details

The bandpass filter removes:

- Low-frequency noise and motion artifacts (below low_cutoff)
- High-frequency noise (above high_cutoff)

Standard EMG bandwidth is 20-450 Hz for surface EMG. The lower cutoff helps remove motion artifacts, while the upper cutoff prevents aliasing and removes high-frequency noise.

Value

Filtered EMG signal, same structure as input.

Examples

```
## Not run:
# Generate sample EMG-like signal
fs <- 1000 # 1000 Hz sampling rate
t <- seq(0, 1, by = 1/fs)
emg <- rnorm(length(t)) + 0.1 * sin(2 * pi * 10 * t) # with low-freq noise

# Apply bandpass filter
filtered <- bandpass_filter(emg, sampling_rate = fs)

## End(Not run)
```

bootstrap_loading_ci *Bootstrap Confidence Intervals for Synergy Loadings*

Description

Computes bootstrap confidence intervals for the loading matrices in mixture PCA or mixture FA models. Uses Procrustes alignment to ensure bootstrap samples are comparable across resamples.

Usage

```
bootstrap_loading_ci(
  list_of_data,
  K,
  r,
  B = 200,
  alpha = 0.05,
  mc_cores = 1,
  model_type = c("MPCA", "MFA"),
  max_iter = 50,
  n_init = 1,
  use_kmeans_init = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

list_of_data	A list of N matrices, each of dimension ($T_i \times M$), containing the observed EMG data.
K	Integer. Number of clusters.
r	Integer. Number of synergies (factors/principal components).
B	Integer. Number of bootstrap iterations. Default: 200.
alpha	Numeric. Significance level for confidence intervals. Default: 0.05 (95% CI).

<code>mc_cores</code>	Integer. Number of cores for parallel computation. Default: 1 (sequential).
<code>model_type</code>	Character. Either "MPCA" or "MFA". Default: "MPCA".
<code>max_iter</code>	Integer. Maximum EM iterations per fit. Default: 50.
<code>n_init</code>	Integer. Number of random initializations. Default: 1.
<code>use_kmeans_init</code>	Logical. Whether to use k-means initialization. Default: TRUE.
<code>seed</code>	Integer. Random seed for reproducibility. Default: NULL.
<code>verbose</code>	Logical. Print progress messages. Default: TRUE.

Details

The bootstrap procedure: 1. Fit the model to the full dataset to obtain reference loadings 2. For each bootstrap iteration $b = 1, \dots, B$: a. Resample subjects with replacement b. Fit the model to the bootstrap sample c. Align bootstrap loadings to reference using Procrustes rotation 3. Compute percentile confidence intervals from aligned samples

Procrustes alignment is essential because factor loadings are only identifiable up to rotation and sign. Without alignment, bootstrap samples would not be comparable, leading to inflated variance estimates.

Value

A list with the following elements:

- loadings_median** List of K matrices ($M \times r$), median loadings for each cluster
- loadings_lower** List of K matrices ($M \times r$), lower CI bound for each cluster
- loadings_upper** List of K matrices ($M \times r$), upper CI bound for each cluster
- loadings_se** List of K matrices ($M \times r$), standard errors for each cluster
- bootstrap_samples** Array of dimension ($B \times K \times M \times r$) containing all aligned bootstrap samples (optional, if `return_samples=TRUE`)
- reference_fit** The reference model fit used for alignment
- n_successful** Number of successful bootstrap iterations
- alpha** The significance level used

Procrustes Alignment

For each cluster k , the bootstrap loading matrix $W_k(b)$ is aligned to the reference W_k using orthogonal Procrustes: $W_{aligned}(b) = W_k(b) R^*$ where R^* minimizes $\|W_k - W_k(b) R\|_F$ subject to $R'R = I$.

The optimal R is found via SVD: if $W_k' W_k(b) = UDV'$, then $R^* = VU'$.

See Also

[mixture_pca_em_fit](#), [mfa_em_fit](#)

Examples

```
## Not run:
# Simulate some data
sim <- simulate_dynamic_synergy_data(N = 50, K = 2, r = 3, M = 8, T_each = 100)

# Compute bootstrap CIs
boot_ci <- bootstrap_loading_ci(
  list_of_data = sim$X_list,
  K = 2,
  r = 3,
  B = 100,
  alpha = 0.05,
  mc_cores = 4
)

# View median loadings for cluster 1
print(boot_ci$loadings_median[[1]])

# View 95% CI width for first synergy, first muscle, cluster 1
width <- boot_ci$loadings_upper[[1]][1,1] - boot_ci$loadings_lower[[1]][1,1]
print(width)

## End(Not run)
```

calc_reconstruction_error_mixtureFA

Compute SSE for a MixtureFA model

Description

Uses the final hard assignments `mfa_fit$z`, reconstruct each subject's data.

Usage

```
calc_reconstruction_error_mixtureFA(list_of_data, mfa_fit)
```

Arguments

<code>list_of_data</code>	The data list.
<code>mfa_fit</code>	The mixture FA result, which contains <code>\$z</code> , <code>\$Lambda[[k]]</code> , <code>\$mu[[k]]</code> , <code>\$Psi[[k]]</code> .

Value

SSE (sum of squared errors).

`calc_reconstruction_error_mixturePCA`*Compute SSE for a MixturePCA model*

Description

Compute SSE for a MixturePCA model

Usage

```
calc_reconstruction_error_mixturePCA(list_of_data, mpca_fit)
```

Arguments

`list_of_data` The data list.

`mpca_fit` The mixture PCA result, containing z and each cluster's P , μ .

Value

SSE (sum of squared errors).

`calc_reconstruction_error_singleFA`*Compute Total Reconstruction SSE for Single FA*

Description

Compute Total Reconstruction SSE for Single FA

Usage

```
calc_reconstruction_error_singleFA(list_of_data, fa_res)
```

Arguments

`list_of_data` The original data list.

`fa_res` A result from `fit_single_factor_analysis`.

Value

A numeric value of the sum of squared errors (SSE).

`calc_reconstruction_error_singlePCA`*Compute Total Reconstruction SSE for Single PCA*

Description

Compute Total Reconstruction SSE for Single PCA

Usage

```
calc_reconstruction_error_singlePCA(list_of_data, pca_res)
```

Arguments

<code>list_of_data</code>	The data list.
<code>pca_res</code>	The result from <code>fit_single_pca</code> .

Value

SSE (sum of squared errors).

`calc_total_SST`*Compute the Total Sum of Squares (SST) for a List of Data Matrices*

Description

This function concatenates all matrices in `list_of_data` into one big matrix, calculates the global mean, and then sums the squared differences from that mean. The result can be used to compute VAF (Variance Accounted For).

Usage

```
calc_total_SST(list_of_data)
```

Arguments

<code>list_of_data</code>	A list of ($T_i \times M$) matrices.
---------------------------	--

Value

A numeric scalar, the total sum of squares (SST).

calibrate_model	<i>Compute Calibration from Model Fit</i>
-----------------	---

Description

Convenience function that extracts responsibilities from a fitted model and computes calibration against true labels.

Usage

```
calibrate_model(fit, z_true, n_bins = 10)
```

Arguments

fit	Fitted mixture model object (MFA or MPCA)
z_true	Integer vector of true cluster labels
n_bins	Number of bins for calibration. Default: 10.

Value

A calibration_result object

Examples

```
## Not run:
# Fit model
fit <- mixture_pca_em_fit(list_of_data, K = 3, r = 2)

# Compute calibration
calib <- calibrate_model(fit, z_true)
print(calib)

## End(Not run)
```

check_future_available	<i>Check if Future Packages are Available</i>
------------------------	---

Description

Checks whether the future ecosystem packages are installed.

Usage

```
check_future_available(batchtools = FALSE)
```

Arguments

batchtools Logical. Also check for future.batchtools.

Value

Logical. TRUE if packages are available.

check_pbs_job_status *Check PBS Job Status*

Description

Checks the status of a PBS job.

Usage

```
check_pbs_job_status(job_id)
```

Arguments

job_id Character. PBS job ID.

Value

List with job status information.

Examples

```
## Not run:
status <- check_pbs_job_status("12345")

## End(Not run)
```

check_SSE_diagnostics *Check SSE Diagnostics for Mixture Models*

Description

This function performs diagnostic checks to investigate SSE (Sum of Squared Errors) behavior in mixture model fits, particularly when SSE increases monotonically with latent dimension r . It examines three key aspects: EM convergence quality, noise variance (Psi) distribution, and VAF-SSE correlation.

Usage

```

check_SSE_diagnostics(
  fit_results,
  list_of_data,
  mixtureFA_fit = NULL,
  mixturePCA_fit = NULL,
  output_dir = NULL,
  psi_small_thresh = 0.01,
  psi_large_thresh = 100
)

```

Arguments

<code>fit_results</code>	A data frame returned by <code>fit_all_methods_on_data</code> , containing columns: Method, BIC, ARI, SSE, VAF. The fitted model objects may be stored as attributes (<code>mixtureFA_fit</code> , <code>mixturePCA_fit</code>) if available.
<code>list_of_data</code>	The original data used for fitting (list of N matrices, each $T_i \times M$). Required for VAF-SSE correlation theoretical line calculation.
<code>mixtureFA_fit</code>	Optional fitted MFA model object. If NULL, will attempt to extract from <code>fit_results</code> attributes. If not available, EM convergence and Psi diagnostics for MFA will be skipped.
<code>mixturePCA_fit</code>	Optional fitted MPCA model object. If NULL, will attempt to extract from <code>fit_results</code> attributes. If not available, EM convergence and Psi diagnostics for MPCA will be skipped.
<code>output_dir</code>	Optional directory path where diagnostic plots will be saved. If NULL (default), plots are displayed on screen only.
<code>psi_small_thresh</code>	Threshold for detecting overfitting (Psi values below this). Default is 0.01.
<code>psi_large_thresh</code>	Threshold for detecting poor fit (Psi values above this). Default is 100.

Details

The function performs three main diagnostic checks:

1. EM Convergence Quality Check

Since the C++ EM implementation doesn't return a convergence trace, this check examines the final model quality by comparing log-likelihoods and checking for degenerate solutions (e.g., empty clusters, extreme parameter values).

2. Psi (Noise Variance) Distribution Check

Examines the distribution of noise variance parameters in both MFA and MPCA models. Extreme values (very small < 0.01 or very large > 100) may indicate:

- Overfitting (Psi approaches 0)
- Poor model fit (Psi approaches infinity)
- Numerical instability

Creates histograms showing the distribution of Psi values across all clusters and muscles.

3. VAF-SSE Correlation Verification

Since $VAF = 1 - SSE/SST$, there should be a strong negative correlation (approximately -1) between VAF and SSE across methods. Deviations from this expected relationship may indicate:

- Scaling inconsistencies in SSE calculation
- Different SST values being used across methods
- Numerical precision issues

Creates a scatter plot of VAF vs SSE with the expected theoretical relationship.

Value

A list with diagnostic results:

convergence_ok Logical; TRUE if convergence appears stable

psi_ok Logical; TRUE if Psi values are in reasonable range

correlation_ok Logical; TRUE if VAF-SSE correlation is strongly negative

warnings Character vector of warning messages

psi_summary_mfa Summary statistics for MFA Psi values

psi_summary_mPCA Summary statistics for MPCA Psi values

vaf_sse_correlation Correlation coefficient between VAF and SSE

Examples

```
## Not run:
# Example 1: Basic usage with fit_all_methods_on_data()
# The fitted models are stored as attributes and automatically extracted
sim_data <- simulate_mixture_data(N = 30, K = 3, r = 3, M = 8, T_each = 200, seed = 123)
fit_results <- fit_all_methods_on_data(
  list_of_data = sim_data$list_of_data,
  z_true = sim_data$z_true,
  N = 30, K = 3, r = 3, M = 8,
  seed = 123
)

# Run diagnostics (fitted models extracted from attributes automatically)
diagnostics <- check_SSE_diagnostics(
  fit_results = fit_results,
  list_of_data = sim_data$list_of_data,
  output_dir = "diagnostics"
)

# Check results
print(diagnostics$warnings)
print(diagnostics$psi_summary_mfa)
print(diagnostics$vaf_sse_correlation)

# Example 2: Pass fitted models explicitly
```

```

# Useful when you have fitted models from other sources
mfa_fit <- mfa_em_fit(sim_data$list_of_data, K = 3, r = 3)
mpca_fit <- mixture_pca_em_fit(sim_data$list_of_data, K = 3, r = 3)

# Create a minimal results data frame
results_df <- data.frame(
  Method = c("MixtureFA", "MixturePCA"),
  SSE = c(100, 120),
  VAF = c(0.85, 0.80)
)

# Run diagnostics with explicit fitted models
diagnostics2 <- check_SSE_diagnostics(
  fit_results = results_df,
  list_of_data = sim_data$list_of_data,
  mixtureFA_fit = mfa_fit,
  mixturePCA_fit = mpca_fit,
  output_dir = "diagnostics2",
  psi_small_thresh = 0.005, # Custom threshold
  psi_large_thresh = 200    # Custom threshold
)

# Example 3: VAF-SSE correlation only (no fitted models)
# Useful for quick checks without EM convergence/Psi diagnostics
diagnostics3 <- check_SSE_diagnostics(
  fit_results = fit_results,
  list_of_data = sim_data$list_of_data
)
# Note: EM convergence and Psi diagnostics will be skipped with a warning

## End(Not run)

```

compare_baselines

Compare Analysis Methods on v2 Dynamic Synergy Data

Description

This function enables pure method-level benchmarking by generating data using only the v2 dynamic synergy simulator and applying all six analysis methods (SingleFA, SinglePCA, TwoStep_FA, TwoStep_PCA, MixtureFA, MixturePCA). This complements [compare_static_vs_dynamic](#) by focusing on method performance rather than model comparison.

Usage

```

compare_baselines(
  N = 50,
  K = 2,
  r = 2,
  M = 6,

```

```

T_each = 100,
cluster_sep_spatial = 1,
cluster_sep_temporal = 1,
cluster_sep_stability = 1,
seed = 123,
n_init = 1,
use_kmeans_init = FALSE,
mc_cores = 1,
method_pca = "EM",
align_basis = TRUE,
refine_assignments = TRUE
)

```

Arguments

N	Number of subjects (time series). Default is 50.
K	Number of clusters. Default is 2.
r	Number of synergies (factor dimension). Default is 2.
M	Number of observed channels (muscles). Default is 6.
T_each	Time-series length per subject. Default is 100.
cluster_sep_spatial	Numeric scalar (0-2) controlling spatial structure differentiation between clusters. Higher values create more distinct spatial modules. Default is 1.0.
cluster_sep_temporal	Numeric scalar (0-2) controlling temporal coordination differentiation between clusters. Higher values create more distinct temporal patterns. Default is 1.0.
cluster_sep_stability	Numeric scalar (0-2) controlling stability-adaptability differentiation between clusters. Higher values create more distinct noise profiles. Default is 1.0.
seed	Random seed for data generation. Default is 123.
n_init	Number of random initializations for mixture methods. Default is 1.
use_kmeans_init	Logical; if TRUE, also try k-means initialization. Default is FALSE.
mc_cores	Number of cores for parallel computation. Default is 1.
method_pca	Method for PCA fitting. Default is "EM".
align_basis	Logical. If TRUE (default), applies Hungarian algorithm alignment in Step 1.5 of TwoStep methods to ensure corresponding components across subjects are in the same order before clustering. Set to FALSE to disable alignment and compare performance with/without this step.
refine_assignments	Logical. If TRUE (default), performs a single reassignment pass after fitting cluster-specific models in Step 4 of TwoStep methods. Each subject is re-assigned to the cluster that minimizes reconstruction error. Set to FALSE to disable reassignment and use only the initial k-means cluster assignments.

Details

This function generates data using [simulate_dynamic_synergy_data](#) with three orthogonal control axes (spatial, temporal, stability), then applies all six analysis methods to evaluate their performance. This is useful for:

- Method benchmarking on realistic dynamic synergy data
- Parameter sensitivity analysis for method performance
- Integration with [run_param_sweep](#) for systematic evaluation

The function complements [compare_static_vs_dynamic](#) by focusing on method performance rather than model comparison. Together, these functions provide comprehensive evaluation of both models and methods.

Value

A data frame with the following columns:

Method	Analysis method name (e.g., "MixtureFA", "SinglePCA")
BIC	Bayesian Information Criterion
ARI	Adjusted Rand Index (clustering accuracy)
SSE	Sum of Squared Errors (reconstruction error)
VAF	Variance Accounted For (1 - SSE/totalSST)
model_type	Always "Dynamic_v2" for this function
seed	Random seed used for data generation
cluster_sep_spatial	Spatial separation parameter
cluster_sep_temporal	Temporal separation parameter
cluster_sep_stability	Stability separation parameter
N	Number of subjects
K	Number of clusters
r	Number of synergies
M	Number of muscles
T_each	Time series length

See Also

[compare_static_vs_dynamic](#) for model comparison, [run_param_sweep](#) for systematic parameter sweep, [simulate_dynamic_synergy_data](#) for data generation

Examples

```
## Not run:
# Basic method comparison on v2 data
results <- compare_baselines(
  N = 50, K = 2, r = 2, M = 6, T_each = 100,
  cluster_sep_spatial = 1.0,
```

```

    cluster_sep_temporal = 1.0,
    cluster_sep_stability = 1.0,
    seed = 123
  )
  print(results)

  # Compare methods with high spatial separation
  results_spatial <- compare_baselines(
    N = 50, K = 2, r = 2, M = 6, T_each = 100,
    cluster_sep_spatial = 2.0,
    cluster_sep_temporal = 0.5,
    cluster_sep_stability = 1.0,
    seed = 456
  )

  # Visualize method performance
  library(ggplot2)
  ggplot(results, aes(x = Method, y = ARI)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    labs(title = "Method Performance on v2 Dynamic Data")

## End(Not run)

```

```
compare_static_vs_dynamic
```

Compare Analysis Methods on Dynamic Synergy Data (v2)

Description

This function uses the v2 dynamic synergy simulator ([simulate_dynamic_synergy_data](#)), which provides three orthogonal control axes: spatial structure, temporal coordination, and stability-adaptability. This allows for systematic evaluation of how different motor control strategies affect muscle synergy structures.

Usage

```

compare_static_vs_dynamic(
  model_types = c("Dynamic_v2"),
  N = 30,
  K = 3,
  r = 3,
  M = 8,
  T_each = 200,
  cluster_sep_spatial = 1,
  cluster_sep_temporal = 1,
  cluster_sep_stability = 1,
  seed = 123,
  output_dir = "results/compare_dynamic_v2",

```

```

    n_init = 1,
    use_kmeans_init = FALSE,
    mc_cores = 1
  )

```

Arguments

<code>model_types</code>	Character vector specifying which model types to use. Default is <code>c("Dynamic_v2")</code> . Only "Dynamic_v2" is supported.
<code>N</code>	Number of subjects (time series). Default is 30.
<code>K</code>	True number of clusters. Default is 3.
<code>r</code>	True factor dimension (number of synergies). Default is 3.
<code>M</code>	Number of observed channels (muscles). Default is 8.
<code>T_each</code>	Time-series length per subject. Default is 200.
<code>cluster_sep_spatial</code>	Numeric scalar (0-2) controlling spatial structure differentiation between clusters. Higher values create more distinct spatial modules. Default is 1.0.
<code>cluster_sep_temporal</code>	Numeric scalar (0-2) controlling temporal coordination differentiation between clusters. Higher values create more distinct temporal patterns. Default is 1.0.
<code>cluster_sep_stability</code>	Numeric scalar (0-2) controlling stability-adaptability differentiation between clusters. Higher values create more distinct noise profiles. Default is 1.0.
<code>seed</code>	Random seed for data generation. Default is 123.
<code>output_dir</code>	Directory path where results will be saved. Default is <code>"results/compare_dynamic_v2"</code> . The directory will be created if it doesn't exist.
<code>n_init</code>	Number of random initializations for mixture methods. Default is 1.
<code>use_kmeans_init</code>	Logical; if TRUE, also try k-means initialization. Default is FALSE.
<code>mc_cores</code>	Number of cores for parallel computation. Default is 1.

Details

The function generates data using the `Dynamic_v2` model and applies six analysis methods (`SingleFA`, `SinglePCA`, `TwoStep_FA`, `TwoStep_PCA`, `MixtureFA`, `MixturePCA`) to evaluate their performance.

The function saves results to an RDS file in the specified output directory with filename format `results_YYYY-MM-DD.rds`.

Data is generated using [simulate_dynamic_synergy_data](#), which creates time-varying synergy activation patterns with three orthogonal control axes: (1) Spatial structure, (2) Temporal coordination, and (3) Stability-Adaptability.

The generated data is analyzed using six methods, allowing comparison of method performance on realistic dynamic synergy data.

Value

A data frame with the following columns:

model_type Character string: "Dynamic_v2"
 Method Analysis method name (e.g., "MixtureFA", "SinglePCA")
 BIC Bayesian Information Criterion
 ARI Adjusted Rand Index (clustering accuracy)
 SSE Sum of Squared Errors (reconstruction error)
 VAF Variance Accounted For (1 - SSE/totalSST)
 seed Random seed used for data generation
 cluster_sep_spatial Spatial separation parameter
 cluster_sep_temporal Temporal separation parameter
 cluster_sep_stability Stability separation parameter
 N Number of subjects
 K Number of clusters
 r Number of synergies
 M Number of muscles
 T_each Time series length

Examples

```
## Not run:
# Basic comparison with v2 simulator
results <- compare_static_vs_dynamic(
  N = 30, K = 3, r = 3, M = 8, T_each = 200,
  cluster_sep_spatial = 1.0,
  cluster_sep_temporal = 1.0,
  cluster_sep_stability = 1.0,
  seed = 123
)
print(results)

# Compare with high spatial separation
results_spatial <- compare_static_vs_dynamic(
  N = 30, K = 3, r = 3, M = 8, T_each = 200,
  cluster_sep_spatial = 2.0,
  cluster_sep_temporal = 0.5,
  cluster_sep_stability = 1.0,
  seed = 456
)

# Visualize results
library(ggplot2)
ggplot(results, aes(x = Method, y = ARI)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Method Performance on Dynamic v2 Data")
```

```
## End(Not run)
```

compute_ari

Compute Adjusted Rand Index (ARI)

Description

Computes the Adjusted Rand Index between true and predicted cluster assignments. This is a NA-safe wrapper around `mclust::adjustedRandIndex`.

Usage

```
compute_ari(true_cluster, pred_cluster)
```

Arguments

`true_cluster` Integer vector of true cluster assignments.
`pred_cluster` Integer vector of predicted cluster assignments.

Details

The Adjusted Rand Index (ARI) measures the similarity between two clusterings, adjusted for chance. It is commonly used to evaluate clustering accuracy when ground truth is available.

Value

Numeric scalar: ARI value in $[-1, 1]$, where 1 indicates perfect agreement. Returns NA if inputs are invalid or `mclust` is not available.

Examples

```
## Not run:
true_cluster <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
pred_cluster <- c(1, 1, 2, 2, 2, 2, 3, 3, 3)
ari <- compute_ari(true_cluster, pred_cluster)

## End(Not run)
```

compute_ari_vaf_table	<i>Compile ARI and VAF into a Single Table</i>
-----------------------	--

Description

This function takes a data frame with a file column (typically from [read_all_results](#)), reads each result file to extract fitted models, computes VAF for MFA and MPCA, and joins these back to the input data frame.

Usage

```
compute_ari_vaf_table(df)
```

Arguments

df A data frame with a file column containing paths to .rds result files.

Value

The input data frame enriched with VAF_MFA and VAF_PCA columns. If models are not available in a file, VAF values will be NA.

Examples

```
## Not run:
df <- read_all_results("results")
df <- compute_ari_vaf_table(df)
head(df)

## End(Not run)
```

compute_BIC	<i>Compute Bayesian Information Criterion (BIC)</i>
-------------	---

Description

Unified interface for computing BIC for both MFA and MPCA models.

Usage

```
compute_BIC(loglik, K, r, M, N_obs, model_type)
```

Arguments

loglik	Numeric scalar, the log-likelihood value
K	Integer, number of clusters
r	Integer, number of factors/synergies
M	Integer, number of variables (muscles)
N_obs	Integer, number of observations (total time points across subjects)
model_type	Character, either "MFA" or "MPCA"

Details

$BIC = -2 * \loglik + p * \log(N_obs)$, where p is the number of parameters. The parameter count differs between MFA and MPCA due to different model structures.

Value

A numeric scalar representing the BIC value (lower is better)

See Also

[compute_BIC_mfa](#), [compute_BIC_mPCA](#)

Examples

```
## Not run:
# Compute BIC for an MFA model
bic <- compute_BIC(loglik, K = 2, r = 3, M = 8, N_obs = 1000, model_type = "MFA")

## End(Not run)
```

compute_BIC_mfa	<i>Compute BIC for an MFA model</i>
-----------------	-------------------------------------

Description

Given the log-likelihood from [compute_logLik_mfa](#), and known K , r , M , N , this function computes $BIC = -2 * \log(L) + nu * \log(N)$, where nu is the parameter count for the MFA model. The corrected formula is $nu = K * (M * r + r * (r-1)/2 + M + M) + (K - 1)$.

Usage

```
compute_BIC_mfa(loglik, K, r, M, N_obs)
```

Arguments

loglik	The total log-likelihood from compute_logLik_mfa.
K	Number of clusters.
r	Factor dimension.
M	Observed dimension (channels).
N_obs	Number of observations (typically sum of all T_i across subjects).

Details

The parameter count includes:

- $M \times r$: factor loadings (Lambda)
- $r \times (r-1)/2$: factor covariances (off-diagonal elements)
- M: unique variances (diagonal Psi)
- M: cluster means (mu)
- $K-1$: mixing proportions

This corrected formula properly penalizes larger r values, improving r estimation stability.

Note: N_obs should typically be the total number of time points (sum of T_i) rather than the number of subjects, as BIC penalizes based on sample size.

Value

A numeric BIC value.

compute_BIC_mPCA	<i>Compute BIC for a Mixture PCA Model</i>
------------------	--

Description

Given the log-likelihood from [compute_logLik_mPCA](#), and the number of clusters K, dimension r, etc., this function computes the corrected BIC: $BIC = -2 * \log(L) + nu * \log(N)$, where nu is the parameter count for MPCA. The corrected formula is $K \times (M \times r + r \times (r-1)/2 + M + 1) + (K-1)$, i.e. for each cluster we have $(M \times r)$ for the directions, $r \times (r-1)/2$ for factor covariances, M for the mean, and 1 for σ^2 . Then $(K-1)$ for mixing proportions.

Usage

```
compute_BIC_mPCA(loglik, K, r, M, N_obs)
```

Arguments

loglik	The total mixture log-likelihood from compute_logLik_mPCA.
K	Number of clusters.
r	Dimension of principal components.
M	Observed dimension.
N_obs	Number of observations (typically sum of all T_i across subjects).

Details

The parameter count includes:

- $M \times r$: loading matrix (W or P)
- $r \times (r-1)/2$: factor covariances (off-diagonal elements)
- M: cluster means (μ)
- 1: isotropic noise variance (σ^2)
- $K-1$: mixing proportions

This corrected formula properly penalizes larger r values, improving r estimation stability.

Note: N_{obs} should typically be the total number of time points (sum of T_i) rather than the number of subjects, as BIC penalizes based on sample size.

Value

A numeric BIC value.

compute_bic_wrapper	<i>Compute BIC for a Fitted Model</i>
---------------------	---------------------------------------

Description

Computes the Bayesian Information Criterion (BIC) for a fitted MFA or MPCA model. This is a unified wrapper that dispatches to the appropriate method-specific function.

Usage

```
compute_bic_wrapper(method, fit, K, r, M, N_obs)
```

Arguments

method	Model type: "MFA" or "MPCA".
fit	A fitted model object.
K	Number of clusters.
r	Factor dimension.
M	Number of observed dimensions (muscles).
N_obs	Number of observations (typically sum of T_i).

Details

BIC is computed as $-2 \log L + p \log(N)$, where L is the likelihood, p is the number of parameters, and N is the sample size.

Value

Numeric scalar: BIC value. Lower is better. Returns NA if computation fails.

Examples

```
## Not run:
fit <- mfa_em_fit(list_of_data, K = 3, r = 2)
N_obs <- sum(sapply(list_of_data, nrow))
M <- ncol(list_of_data[[1]])
bic <- compute_bic_wrapper("MFA", fit, K = 3, r = 2, M = M, N_obs = N_obs)

## End(Not run)
```

compute_calibration *Compute Calibration Statistics for Mixture Model*

Description

Computes calibration statistics by binning predicted probabilities and comparing to observed frequencies. This helps diagnose whether the model's probabilistic predictions are reliable.

Usage

```
compute_calibration(resp, z_true, n_bins = 10)
```

Arguments

resp	Matrix of responsibilities (N x K), where resp[i,k] is the posterior probability that subject i belongs to cluster k. Rows should sum to 1.
z_true	Integer vector of length N with true cluster labels (1 to K). If not available, can use hard cluster assignments from another method.
n_bins	Integer. Number of bins for calibration curve. Default: 10.

Details

Calibration is computed for each cluster separately: 1. Bin the predicted probabilities into n_bins equal-width bins 2. For each bin, compute mean predicted probability and observed frequency 3. A well-calibrated model has mean_predicted close to mean_observed

The Expected Calibration Error (ECE) summarizes overall calibration:

$$ECE = \sum_{b=1}^B \frac{|S_b|}{N} |acc(S_b) - conf(S_b)|$$

where S_b is the set of samples in bin b, acc is accuracy (observed frequency), and conf is confidence (mean predicted probability).

Value

A list with class "calibration_result" containing:

calibration_df Data frame with columns: bin_midpoint, mean_predicted, mean_observed, n_samples, cluster

ece Expected Calibration Error (weighted average bin error)

mce Maximum Calibration Error (worst bin)

n_bins Number of bins used

K Number of clusters

N Number of subjects

See Also

[plot_calibration](#), [compute_ece](#)

Examples

```
## Not run:
# Fit a mixture model
fit <- mixture_pca_em_fit(list_of_data, K = 3, r = 2)

# Compute calibration using true labels
calib <- compute_calibration(fit$res, z_true)

# View ECE
print(calib$ece)

# Plot calibration curve
plot_calibration(calib)

## End(Not run)
```

compute_cluster_sizes *Compute Cluster Sizes from a Fitted MFA (or Mixture PCA) Model*

Description

Given a result with a vector z of length N (the cluster assignments), this function returns a frequency table, i.e., how many subjects are assigned to each cluster.

Usage

```
compute_cluster_sizes(res)
```

Arguments

res A fitted mixture model object that contains z .

Value

A table of cluster frequencies.

Examples

```
## Not run:
tab <- compute_cluster_sizes(fit_mfa)
print(tab)

## End(Not run)
```

```
compute_cluster_sizes_mPCA
```

Compute Cluster Sizes for Mixture PCA

Description

Like [compute_cluster_sizes](#), but for a Mixture PCA model result.

Usage

```
compute_cluster_sizes_mPCA(res)
```

Arguments

`res` A fitted mixture PCA model result, containing `$z`.

Value

A table of frequencies of cluster assignments.

```
compute_clusterwise_subspace_score
```

Compute Cluster-wise Subspace Recovery Score

Description

Computes subspace similarity scores for each cluster, using the aligned mapping between estimated and true clusters. This provides a detailed assessment of how well each cluster's subspace is recovered.

Usage

```
compute_clusterwise_subspace_score(true_Lambda_list, est_Lambda_list, mapping)
```

Arguments

<code>true_Lambda_list</code>	List of K_{true} matrices, each $(M \times r)$ representing true cluster-specific basis matrices.
<code>est_Lambda_list</code>	List of K_{hat} matrices, each $(M \times r_{\text{hat}})$ representing estimated cluster-specific basis matrices.
<code>mapping</code>	Named integer vector from align_clusters_to_truth , mapping estimated cluster labels to true cluster labels.

Details

For each true cluster k :

1. Find the estimated cluster that maps to k (using the mapping)
2. Compute subspace similarity between `true_Lambda_list[[k]]` and the corresponding estimated basis
3. If no estimated cluster maps to k , the score is NA

This function is NA-safe: if estimation failed for some clusters (NULL or invalid matrices), those clusters receive NA scores and are excluded from the mean calculation.

Value

A list with:

`scores` Named numeric vector of subspace similarity scores for each true cluster (names are cluster indices as characters).

`mean_score` Numeric scalar: mean of non-NA scores.

`n_matched` Integer: number of true clusters with a valid estimated match.

Returns a list with NA values if inputs are invalid.

Examples

```
## Not run:
# Generate data and fit model
data <- generate_synergy_data(N = 30, K = 3, r = 2, seed = 123)
fit <- mfa_em_fit(data$list_of_data, K = 3, r = 2)

# Align clusters
alignment <- align_clusters_to_truth(data$true_cluster, fit$z)

# Compute cluster-wise scores
scores <- compute_clusterwise_subspace_score(
  data$true_Lambda_list,
  fit$Lambda,
  alignment$mapping
)
```

```
## End(Not run)
```

`compute_ece`*Compute Expected Calibration Error*

Description

A convenience function to compute only the ECE from responsibilities and true labels.

Usage

```
compute_ece(resp, z_true, n_bins = 10)
```

Arguments

<code>resp</code>	Matrix of responsibilities (N x K)
<code>z_true</code>	Integer vector of true cluster labels (length N)
<code>n_bins</code>	Number of bins. Default: 10.

Details

ECE measures the average absolute difference between predicted probabilities and observed frequencies. A value of 0 indicates perfect calibration; values above 0.1 suggest significant miscalibration.

Guidelines for interpretation:

- ECE < 0.02: Excellent calibration
- ECE 0.02-0.05: Good calibration
- ECE 0.05-0.10: Moderate calibration
- ECE > 0.10: Poor calibration

Value

Numeric scalar, the ECE value. Lower is better (0 = perfect calibration).

See Also

[compute_calibration](#)

Examples

```
## Not run:
ece <- compute_ece(fit$resp, z_true)
cat(sprintf("ECE: %.3f\n", ece))

## End(Not run)
```

compute_entropy

Compute Classification Entropy from Responsibilities

Description

Computes the entropy term used in ICL (Integrated Complete-data Likelihood). Higher entropy indicates more uncertain cluster assignments.

Usage

```
compute_entropy(resp)
```

Arguments

resp An (N x K) matrix of responsibilities (posterior probabilities), where resp[i,k] is the probability that subject i belongs to cluster k.

Details

The entropy is computed as:

$$H = - \sum_{i=1}^N \sum_{k=1}^K r_{ik} \log(r_{ik})$$

where r_{ik} is the responsibility of subject i for cluster k. A small constant (1e-16) is added to avoid log(0).

Value

A numeric scalar representing the entropy: `-sum(resp * log(resp))`.

Examples

```
## Not run:
# Responsibilities from MFA fit
entropy <- compute_entropy(fit_mfa$resp)

## End(Not run)
```

`compute_factor_scores` *Compute Factor Scores for a Single MFA Cluster*

Description

Given a single cluster's parameters (μ , Λ , Ψ) and data X , this function returns the estimated factor scores.

Usage

```
compute_factor_scores(X, mu, Lambda, Psi)
```

Arguments

X	A numeric matrix of size $(T \times M)$.
μ	A length- M mean vector.
Λ	An $M \times r$ loading matrix.
Ψ	An $M \times M$ diagonal matrix for unique variances.

Details

This follows the standard posterior mean formula in factor analysis: $E[Z|X] = (I + \Lambda^T \Psi^{-1} \Lambda)^{-1} \Lambda^T \Psi^{-1} (X - \mu)$.

Value

A $(T \times r)$ matrix of factor scores.

See Also

[compute_global_vaf_mfa](#)

Examples

```
## Not run:
# Suppose we have X (T x M), mu (M), Lambda (M x r), Psi (M x M).
scores <- compute_factor_scores(X, mu, Lambda, Psi)
head(scores)

## End(Not run)
```

```
compute_factor_scores_mfa
```

Compute Factor Scores for All Subjects in an MFA Model

Description

Given a fitted MFA model (with z , μ , Λ , Ψ for each cluster), this function computes the posterior factor scores $Z_i(t)$ for each subject i and stores them in `model$factor_scores[[i]]`.

Usage

```
compute_factor_scores_mfa(mfa_fit, list_of_data)
```

Arguments

<code>mfa_fit</code>	A fitted MFA model, typically from <code>mfa_em_fit()</code> or <code>select_optimal_K_r_mfa()</code> \$best_model_info\$containing: • <code>z[i]</code>: cluster assignments, • <code>Lambda[[k]]</code>, <code>Psi[[k]]</code>, <code>mu[[k]]</code> for each cluster k.
<code>list_of_data</code>	The same list of data ($T_i \times M$) used to fit the model.

Details

Internally, it calls [compute_factor_scores_optimized](#) for each subject, using that subject's assigned cluster $k = z[i]$.

Value

The same `mfa_fit` object, but with a new element `factor_scores` (a list of length N), where each `factor_scores[[i]]` is a ($T_i \times r$) matrix of factor scores for subject i .

Examples

```
## Not run:
mfa_fit <- compute_factor_scores_mfa(mfa_fit, list_of_data)
# Now mfa_fit$factor_scores[[i]] is available for each subject i

## End(Not run)
```

compute_factor_scores_mpca

Compute PC Factor Scores for All Subjects in a Mixture PCA Model

Description

Given a fitted MPCA model (with z , μ , W , σ^2 for each cluster), this function computes the posterior factor (PC) scores $Z_i(t)$ for each subject i and stores them in `mpca_fit$factor_scores[[i]]`.

Usage

```
compute_factor_scores_mpca(mpca_fit, list_of_data)
```

Arguments

`mpca_fit` A fitted MPCA model from `mixture_pca_em_fit()` or `select_optimal_K_r_mpca()`\$best_model_in typically containing:

- `z[i]`: cluster assignments (1..K),
- `mu[[k]]`, `W[[k]]`, `sigma2[k]` for each cluster,
- possibly `P[[k]]`, `D[[k]]` instead of `W[[k]]`.

`list_of_data` The same list of data ($T_i \times M$) used to fit the model.

Details

We use [compute_factor_scores_mpca_single](#) for each subject, with the appropriate cluster parameters. This assumes isotropic noise `sigma2[k]` for cluster k .

Value

The same `mpca_fit` object, but with a new element `factor_scores`, where `factor_scores[[i]]` is a ($T_i \times r$) matrix of PC scores for subject i .

Examples

```
## Not run:
mpca_fit <- compute_factor_scores_mpca(mpca_fit, list_of_data)
mpca_fit$factor_scores[[1]] # (T_1 x r) PC scores

## End(Not run)
```

```
compute_factor_scores_mpca_single
```

Compute Factor (PC) Scores for a Single Subject in a Mixture PCA Cluster

Description

Given a single subject's data X , and the cluster parameters (μ , W , σ^2), this function computes the posterior factor scores ($T \times r$) under a PPCA assumption with isotropic noise $\sigma^2 * I$.

Usage

```
compute_factor_scores_mpca_single(X, mu, W, sigma2)
```

Arguments

X	A ($T \times M$) data matrix for one subject/time-series.
μ	A length- M mean vector.
W	An ($M \times r$) loading (projection) matrix for the cluster.
σ^2	A scalar representing isotropic noise variance (σ^2).

Details

We use the posterior mean formula for PPCA with isotropic noise: $z(t) = \text{inv}(W^T W + \sigma^2 * I_r) * W^T * (x(t) - \mu)$. This is applied row by row in X , but we can do it in a vectorized manner.

If your noise model is not purely $\sigma^2 * I$, you will need a more general approach. Also note that if $T=1$, the result might be a ($1 \times r$) matrix or (r)-vector, so be sure to keep it as a matrix if desired.

Value

A ($T \times r$) matrix of factor (PC) scores for this subject.

```
compute_factor_scores_optimized
```

Compute Factor Scores for a Single Subject in MFA

Description

Given a single subject's data X and the cluster's parameters (μ , Λ , Ψ), this function computes the posterior factor scores (of size $T \times r$) in a standard Factor Analysis approach.

Usage

```
compute_factor_scores_optimized(X, mu, Lambda, Psi)
```

Arguments

<code>X</code>	A (T x M) data matrix for one subject/time-series.
<code>mu</code>	A length-M mean vector for the cluster.
<code>Lambda</code>	An (M x r) factor loading matrix for the cluster.
<code>Psi</code>	An (M x M) diagonal covariance matrix (unique variances).

Details

We use the standard formula for Factor Analysis posterior means:

$$\mathbf{Z}_i(t) = (\mathbf{I}_r + \Lambda^\top \Psi^{-1} \Lambda)^{-1} \Lambda^\top \Psi^{-1} [\mathbf{x}_i(t) - \mu].$$

The code is implemented in a vectorized manner. If $r = \text{ncol}(\Lambda)$, then the output dimension is (T x r).

Value

A (T x r) matrix of factor scores $\mathbf{Z}_i(t)$, where $t = 1 \dots T$.

Examples

```
## Not run:
X_i <- list_of_data[[i]] # (T x M)
# from model$mu[[k]], model$Lambda[[k]], model$Psi[[k]]
Z_i <- compute_factor_scores_optimized(X_i, mu_k, Lambda_k, Psi_k)
dim(Z_i) # (T x r)

## End(Not run)
```

```
compute_global_vaf_mfa
```

Compute Global VAF (1 - SSE/SST) for an MFA Model

Description

This function takes a fitted MFA model (with elements `z`, `mu`, `Lambda`, `Psi`) and reconstructs all subjects' data from the factor scores, computing the ratio of explained variance (1 - SSE/SST).

Usage

```
compute_global_vaf_mfa(list_of_data, res)
```

Arguments

- `list_of_data` A list of matrices, each ($T_i \times M$), containing the original data.
- `res` A fitted MFA model result. It must contain:
- `z`: A length- N vector of cluster assignments ($1..K$).
 - `mu`: A list of length K , each a mean vector of length M .
 - `Lambda`: A list of length K , each an $M \times r$ loading matrix.
 - `Psi`: A list of length K , each an $M \times M$ diagonal matrix.

Details

For each subject i , we identify its cluster assignment k_i , compute factor scores via [compute_factor_scores](#), reconstruct $\hat{X}$ from the factor model, accumulate the sum of squares of residuals (SSE), and the sum of squares total (SST).

Value

A numeric scalar for the global variance accounted for, $1 - SSE/SST$.

See Also

[compute_factor_scores](#)

Examples

```
## Not run:
gvaf <- compute_global_vaf_mfa(my_data_list, fit_mfa)
cat("Global VAF:", gvaf, "\n")

## End(Not run)
```

compute_global_vaf_mPCA

Compute Global VAF ($1 - SSE/SST$) for Mixture PCA

Description

Similar to [compute_global_vaf_mfa](#), but uses a Mixture PCA model structure. Given a model with `z`, `mu`, `P`, `D`, `Psi`, we reconstruct each subject's data from the principal component scores, accumulate SSE, and compare to total sum of squares (SST).

Usage

```
compute_global_vaf_mPCA(list_of_data, res)
```

Arguments

- list_of_data

A list of ($T_i \times M$) data matrices.
- res

The fitted Mixture PCA model, typically from `mixture_pca_em_fit`. Must contain:
 - `z`: cluster assignments (length N).
 - `mu`: list of length K , each (M) vector.
 - `P`: list of length K , each ($M \times r$).
 - `D`: list of length K , each ($r \times r$) diagonal.
 - `Psi`: list of length K , each ($M \times M$) diagonal.

Details

The user must define how to reconstruct data from PCA parameters. Typically, for each cluster k , we have $W = P * D$, so we either compute posterior scores or do a direct projection.

Value

A numeric scalar, $1 - SSE/SST$ across all subjects and time steps.

compute_ICL	<i>Compute ICL for a Mixture Model (Unified Interface)</i>
-------------	--

Description

Unified interface for computing ICL (Integrated Complete-data Likelihood) for either MFA or MPCA models.

Usage

```
compute_ICL(loglik, K, r, M, N_obs, resp, model_type)
```

Arguments

- loglik

The total log-likelihood from the fitted model.
- K

Number of clusters.
- r

Factor/principal component dimension.
- M

Observed dimension (channels).
- N_obs

Number of observations.
- resp

An ($N \times K$) matrix of responsibilities.
- model_type

Either "MFA" or "MPCA".

Value

A numeric ICL value.

See Also

[compute_ICL_mfa](#), [compute_ICL_mPCA](#)

compute_ICL_mfa	<i>Compute ICL for a Mixture Factor Analysis Model</i>
-----------------	--

Description

ICL (Integrated Complete-data Likelihood) is a model selection criterion that penalizes uncertain cluster assignments more heavily than BIC. This makes it more conservative for selecting the number of clusters K.

Usage

```
compute_ICL_mfa(loglik, K, r, M, N_obs, resp)
```

Arguments

loglik	The total log-likelihood from the MFA model.
K	Number of clusters.
r	Factor dimension.
M	Observed dimension (channels).
N_obs	Number of observations (typically sum of all T_i across subjects).
resp	An (N x K) matrix of responsibilities (posterior probabilities).

Details

ICL is computed as:

$$ICL = BIC + 2 \times Entropy$$

where Entropy = -sum(resp * log(resp)).

The entropy penalty discourages models where cluster assignments are uncertain, which typically happens when K is overestimated (extra clusters with low membership). This makes ICL more conservative than BIC for K selection.

Value

A numeric ICL value.

References

Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7), 719-725.

See Also

[compute_BIC_mfa](#), [compute_entropy](#)

Examples

```
## Not run:
icl <- compute_ICL_mfa(fit$logLik, K = 2, r = 3, M = 8,
                      N_obs = 1000, resp = fit$resp)

## End(Not run)
```

compute_ICL_mPCA

Compute ICL for a Mixture PCA Model

Description

ICL (Integrated Complete-data Likelihood) is a model selection criterion that penalizes uncertain cluster assignments more heavily than BIC. This makes it more conservative for selecting the number of clusters K .

Usage

```
compute_ICL_mPCA(loglik, K, r, M, N_obs, resp)
```

Arguments

loglik	The total log-likelihood from the MPCA model.
K	Number of clusters.
r	Principal component dimension.
M	Observed dimension (channels).
N_obs	Number of observations (typically sum of all T_i across subjects).
resp	An ($N \times K$) matrix of responsibilities (posterior probabilities).

Details

ICL is computed as:

$$ICL = BIC + 2 \times Entropy$$

where Entropy = $-\sum(\text{resp} * \log(\text{resp}))$.

Value

A numeric ICL value.

See Also

[compute_BIC_mPCA](#), [compute_entropy](#)

compute_logLik	<i>Compute Log-Likelihood for Mixture Models</i>
----------------	--

Description

Unified interface for computing log-likelihood for both MFA and MPCA models. Automatically detects model type from the fit object structure.

Usage

```
compute_logLik(list_of_data, fit, model_type = NULL)
```

Arguments

list_of_data	A list of N matrices, each of dimension (T _i x M)
fit	A fitted model object (MFA or MPCA)
model_type	Character, either "MFA" or "MPCA". If NULL, auto-detected from the fit object structure.

Value

A numeric scalar representing the total log-likelihood

See Also

[compute_logLik_mfa](#), [compute_logLik_mPCA](#)

Examples

```
## Not run:  
# Works with either MFA or MPCA fit objects  
ll <- compute_logLik(list_of_data, fit)  
  
# Or explicitly specify model type  
ll <- compute_logLik(list_of_data, fit, model_type = "MFA")  
  
## End(Not run)
```

compute_logLik_mfa	<i>Compute the Mixture-Model Log-Likelihood for MFA</i>
--------------------	---

Description

Given a fitted MFA model (with K clusters, each containing $\text{Lambda}[[k]]$, $\mu[[k]]$, $\text{Psi}[[k]]$, and mixing proportion $\pi[k]$), this function computes the standard mixture log-likelihood:

$$\sum_{i=1}^N \log \left(\sum_{k=1}^K \pi_k \prod_{t=1}^{T_i} \mathcal{N}(x_{i,t} \mid \mu_k, \Sigma_k) \right),$$

where $\text{Sigma}_k = \text{Lambda}_k * \text{Lambda}_k^T + \text{Psi}_k$.

Usage

```
compute_logLik_mfa(list_of_data, mfa_fit)
```

Arguments

- | | |
|---------------------------|---|
| <code>list_of_data</code> | A list of length N, each an ($T_i \times M$) matrix. |
| <code>mfa_fit</code> | A fitted MFA model, containing: <ul style="list-style-type: none"> • K clusters (implicit from <code>length(mfa_fit\$Lambda)</code>), • $\text{Lambda}[[k]]$, $\mu[[k]]$, $\text{Psi}[[k]]$, • π as a length-K vector of mixing proportions. |

Details

Internally uses rowwise multivariate normal densities. You will need `mvtnorm::dmvnorm` or an equivalent. Ensure each Sigma_k is well-defined and invertible if needed.

Value

A numeric scalar, the total log-likelihood.

compute_logLik_mPCA	<i>Compute the Mixture-Model Log-Likelihood for Mixture PCA</i>
---------------------	---

Description

Given a fitted Mixture PCA model with $\text{P}[[k]]$, $\mu[[k]]$, $\text{sigma2}[k]$, and $\pi[k]$, we compute

$$\sum_{i=1}^N \log \left(\sum_{k=1}^K \pi_k \prod_{t=1}^{T_i} \mathcal{N}(x_{i,t} \mid \mu_k, \Sigma_k) \right)$$

where $\text{Sigma}_k = \text{W}_k * \text{W}_k^T + \text{sigma2}_k * \text{I}$. A typical approach is to reconstruct Sigma_k from $\text{P}[[k]]$ if needed (or $\text{W}[[k]]$).

Usage

```
compute_logLik_mpca(list_of_data, mpca_fit)
```

Arguments

`list_of_data` A list of length N, each (T_i × M).

`mpca_fit` A fitted Mixture PCA model, containing:

- P[[k]] or W[[k]] for cluster k,
- mu[[k]], sigma2[k], pi[k].

Details

We assume each cluster k has $\Sigma_k = W_k * W_k^T + \sigma^2_k * I$. You must assemble that to call `mvtnorm::dmvnorm` row by row.

Value

A numeric scalar, the total mixture log-likelihood.

`compute_mvc_reference` *Compute MVC Reference Values*

Description

Computes Maximum Voluntary Contraction (MVC) reference values from MVC trial data for use in EMG normalization.

Usage

```
compute_mvc_reference(
  mvc_data,
  method = "peak",
  window_ms = 500,
  top_percent = 5,
  sampling_rate = NULL
)
```

Arguments

`mvc_data` MVC trial data. Can be a matrix (T × M) or a list of matrices (multiple MVC trials).

`method` Method to compute MVC:

- "peak": Maximum value (default)
- "mean_peak": Mean of top N
- "rms_peak": RMS of the peak window

window_ms	Window size in ms for peak detection (used with "rms_peak"). Default is 500 ms.
top_percent	Percentage of top values to average (used with "mean_peak"). Default is 5.
sampling_rate	Sampling rate in Hz (required for "rms_peak").

Value

Numeric vector of MVC values, one per channel.

Examples

```
# From single MVC trial
mvc_trial <- matrix(abs(rnorm(1000 * 4)), ncol = 4)
mvc_ref <- compute_mvc_reference(mvc_trial)

# From multiple MVC trials (takes maximum across trials)
mvc_trials <- list(
  trial1 = matrix(abs(rnorm(1000 * 4)), ncol = 4),
  trial2 = matrix(abs(rnorm(1000 * 4)), ncol = 4)
)
mvc_ref <- compute_mvc_reference(mvc_trials)
```

compute_reconstruction_error

Compute Reconstruction Error

Description

Unified interface for computing reconstruction error for both MFA and MPCA models. Automatically detects model type from the fit object structure.

Usage

```
compute_reconstruction_error(
  list_of_data,
  fit,
  model_type = NULL,
  metric = c("sse", "rmse")
)
```

Arguments

list_of_data	A list of N matrices, each of dimension (T _i x M)
fit	A fitted model object (MFA or MPCA)
model_type	Character, either "MFA" or "MPCA". If NULL, auto-detected from the fit object structure.
metric	Character, one of "sse" (sum of squared errors) or "rmse" (root mean squared error). Default: "sse"

Value

A numeric scalar representing the reconstruction error

Examples

```
## Not run:
# Compute SSE
sse <- compute_reconstruction_error(list_of_data, fit)

# Compute RMSE
rmse <- compute_reconstruction_error(list_of_data, fit, metric = "rmse")

## End(Not run)
```

compute_smoothness_penalty

Compute Smoothness Penalty for Factor Scores

Description

Computes the temporal smoothness penalty for factor scores using second-order differences. This penalty measures how "wiggly" the factor activation profiles are, with higher values indicating more high-frequency noise.

Usage

```
compute_smoothness_penalty(factor_scores)
```

Arguments

factor_scores A list of N matrices, each (T_i x r), containing factor scores for each subject.

Details

The smoothness penalty is computed as:

$$\text{smoothness} = \sum_{i=1}^N \sum_{j=1}^r \sum_{t=3}^{T_i} (z_{i,j,t} - 2z_{i,j,t-1} + z_{i,j,t-2})^2$$

where $z_{i,j,t}$ is the j-th factor score for subject i at time t.

This is equivalent to summing the squared second-order differences across all time series. When r is too large, models tend to fit high-frequency noise, producing wiggly factors with high smoothness penalties.

Value

A numeric scalar representing the total smoothness penalty across all subjects and factors.

Examples

```
## Not run:
# Compute factor scores first
mfa_fit <- compute_factor_scores_mfa(mfa_fit, list_of_data)
# Then compute smoothness
smooth_val <- compute_smoothness_penalty(mfa_fit$factor_scores)

## End(Not run)
```

compute_subspace_similarity

Compute Subspace Similarity Between Two Basis Matrices

Description

Computes the similarity between two subspaces spanned by basis matrices using QR decomposition and SVD. This metric measures how well the estimated subspace recovers the true subspace, independent of rotation and scaling.

Usage

```
compute_subspace_similarity(L_true, L_hat)
```

Arguments

L_true	A (M x r_true) matrix representing the true basis.
L_hat	A (M x r_hat) matrix representing the estimated basis.

Details

The subspace similarity is computed as follows:

1. Orthonormalize both bases using QR decomposition
2. Compute the SVD of $Q_true' * Q_hat$
3. Return the mean of squared singular values (up to min_dim)

This metric is invariant to rotation and scaling of the basis vectors, making it suitable for comparing factor analysis solutions where rotation indeterminacy exists.

When $r_true \neq r_hat$, the metric uses $\min(r_true, r_hat)$ dimensions, which allows comparison even when the estimated rank differs from truth.

Value

Numeric scalar in $[0, 1]$, where 1 indicates identical subspaces. Returns NA if inputs are invalid (NULL, empty, or contain NA/NaN/Inf).

Examples

```
## Not run:
# Same subspace (rotated)
L_true <- matrix(c(1, 0, 0, 1), 2, 2)
R <- matrix(c(cos(pi/4), -sin(pi/4), sin(pi/4), cos(pi/4)), 2, 2)
L_hat <- L_true %*% R
compute_subspace_similarity(L_true, L_hat) # Should be ~1

# Orthogonal subspaces
L_true <- matrix(c(1, 0, 0, 0), 2, 2)
L_hat <- matrix(c(0, 0, 0, 1), 2, 2)
compute_subspace_similarity(L_true, L_hat) # Should be ~0

## End(Not run)
```

compute_VAF

Compute Global Variance Accounted For (VAF)

Description

Unified interface for computing global VAF for both MFA and MPCA models. Automatically detects model type from the fit object structure.

Usage

```
compute_VAF(list_of_data, fit, model_type = NULL)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each of dimension ($T_i \times M$)
<code>fit</code>	A fitted model object (MFA or MPCA)
<code>model_type</code>	Character, either "MFA" or "MPCA". If NULL, auto-detected from the fit object structure.

Details

VAF is computed as $1 - SSE/SST$, where SSE is the sum of squared errors between observed and reconstructed data, and SST is the total sum of squares.

Value

A numeric scalar representing the global VAF (0 to 1)

See Also

[compute_global_vaf_mfa](#), [compute_global_vaf_mPCA](#)

Examples

```
## Not run:
# Works with either MFA or MPCA fit objects
vaf <- compute_VAF(list_of_data, fit)

# Or explicitly specify model type
vaf <- compute_VAF(list_of_data, fit, model_type = "MFA")

## End(Not run)
```

compute_VAF_mfa

*Compute VAF for a Fitted MFA Model***Description**

This function computes the Variance Accounted For (VAF) for a fitted MFA model across all subjects, reconstructing each subject's data via factor scores and summing squared errors.

Usage

```
compute_VAF_mfa(list_of_data, mfa_fit, z)
```

Arguments

list_of_data	A list of subject data matrices, each ($T_i \times M$).
mfa_fit	An MFA fit object containing Lambda, Psi, mu, etc.
z	An integer vector of length N specifying each subject's cluster ID (1..K).

Value

A numeric scalar representing the total VAF over all subjects.

Examples

```
## Not run:
sim <- simulate_mixture_data(N = 50, K = 3, r = 2, M = 6, T_each = 100)
fit <- mfa_em_fit(sim$list_of_data, K = 3, r = 2)
vaf <- compute_VAF_mfa(sim$list_of_data, fit, fit$z)
print(vaf)

## End(Not run)
```

compute_VAF_mpca	<i>Compute VAF for a Mixture PCA (PPCA) Model</i>
------------------	---

Description

This function computes the Variance Accounted For (VAF) for a fitted Mixture PCA model.

Usage

```
compute_VAF_mpca(list_of_data, mpca_fit, z)
```

Arguments

<code>list_of_data</code>	A list of subject data matrices.
<code>mpca_fit</code>	A fitted MPCA model with P, D, Psi, mu.
<code>z</code>	An integer vector of cluster assignments.

Value

A numeric scalar for the overall VAF.

Examples

```
## Not run:
sim <- simulate_mixture_data(N = 50, K = 3, r = 2, M = 6, T_each = 100)
fit <- mixture_pca_em_fit(sim$list_of_data, K = 3, r = 2)
vaf <- compute_VAF_mpca(sim$list_of_data, fit, fit$z)
print(vaf)

## End(Not run)
```

compute_vaf_wrapper	<i>Compute VAF (Variance Accounted For) for a Fitted Model</i>
---------------------	--

Description

Computes the global Variance Accounted For (VAF) for a fitted MFA or MPCA model. This is a unified wrapper that dispatches to the appropriate method-specific function.

Usage

```
compute_vaf_wrapper(method, list_of_data, fit)
```

Arguments

method	Model type: "MFA" or "MPCA".
list_of_data	A list of data matrices, each (T _i x M).
fit	A fitted model object.

Details

VAF is computed as $1 - SSE/SST$, where SSE is the sum of squared reconstruction errors and SST is the total sum of squares.

Value

Numeric scalar: VAF value in [0, 1], where 1 indicates perfect reconstruction. Returns NA if computation fails.

Examples

```
## Not run:
fit <- mfa_em_fit(list_of_data, K = 3, r = 2)
vaf <- compute_vaf_wrapper("MFA", list_of_data, fit)

## End(Not run)
```

compute_vss_criterion *Compute VSS (Very Simple Structure) Criterion for Factor Loadings*

Description

Computes the Very Simple Structure (VSS) criterion, which evaluates how well factor loadings exhibit simple structure (each variable loads strongly on one factor, weakly on others).

Usage

```
compute_vss_criterion(
  Lambda,
  threshold = 0.5,
  method = c("threshold", "ratio"),
  rotate = c("none", "varimax")
)
```

Arguments

<code>Lambda</code>	A list of K loading matrices, each (M x r), representing factor loadings for each cluster.
<code>threshold</code>	Threshold for determining "strong" vs "weak" loadings. Default is 0.5. Loadings with absolute value above this threshold are considered strong.
<code>method</code>	Method for computing VSS. Options: "threshold" (count variables with exactly one strong loading above threshold), "ratio" (compute concentration ratio $\max(\lambda^2)/\sum(\lambda^2)$ for each variable). Default is "threshold".
<code>rotate</code>	Rotation method to apply before computing VSS. Options: "none" (no rotation), "varimax" (orthogonal rotation). Default is "none". VSS on unrotated FA loadings is rotation-dependent and often uninformative. Rotation is recommended for meaningful VSS.

Details

The VSS criterion evaluates simple structure in factor loadings. Two methods are available:

****Threshold method (default):**** For each variable (muscle), count how many factors have $|\text{loading}| > \text{threshold}$. Simple structure is achieved when this count is exactly 1. The VSS score is the proportion of variables across all clusters that exhibit simple structure.

****Ratio method:**** For each variable (muscle), compute the concentration ratio $\max(\lambda^2)/\sum(\lambda^2)$, which measures how concentrated the loadings are on a single factor. The VSS score is the average concentration ratio across all variables. This method is more robust when loadings are small in absolute magnitude, as it doesn't depend on an arbitrary threshold.

This implementation is inspired by the `psych::VSS()` function, adapted for mixture models where each cluster has its own loading matrix.

Value

A numeric scalar representing the VSS criterion. Higher values indicate better simple structure. Returns NA if rotation fails or loadings are degenerate.

References

Revelle, W., & Rocklin, T. (1979). Very Simple Structure: An Alternative Procedure For Estimating The Optimal Number Of Interpretable Factors. *Multivariate Behavioral Research*, 14(4), 403-414.

Examples

```
## Not run:
# For an MFA model
vss_val <- compute_vss_criterion(mfa_fit$Lambda, threshold = 0.5)

## End(Not run)
```

count_completed_tasks	<i>Count Completed PBS Tasks</i>
-----------------------	----------------------------------

Description

Counts how many tasks have completed by checking partial result files.

Usage

```
count_completed_tasks(output_dir, n_total = NULL)
```

Arguments

- | | |
|------------|---|
| output_dir | Character. Output directory. |
| n_total | Integer. Total number of expected tasks (optional). |

Value

List with completion statistics.

emg_preset_config	<i>EMG Preprocessing Preset Configurations</i>
-------------------	--

Description

Returns preset configurations for common EMG preprocessing scenarios.

Usage

```
emg_preset_config(preset = "default")
```

Arguments

- | | |
|--------|--|
| preset | Preset name: <ul style="list-style-type: none">• "default": General-purpose EMG preprocessing• "gait": Optimized for gait/walking analysis• "isometric": Optimized for isometric contractions• "fine_motor": Optimized for fine motor tasks |
|--------|--|

Value

A list containing preprocessing configuration for each step.

Examples

```
# Get default configuration
config <- emg_preset_config("default")

# Modify for custom needs
config$notch$freq <- 60 # Change to 60 Hz
config$envelope$cutoff <- 10 # Higher cutoff for faster movements
```

```
evaluate_model_selection
```

Evaluate Model Selection Results

Description

Comprehensive evaluation of model selection results, computing all relevant metrics including ARI, VAF, BIC, logLik, and selection accuracy.

Usage

```
evaluate_model_selection(
  list_of_data,
  true_cluster,
  K_true,
  r_true,
  select_result,
  runtime_sec = NA_real_
)
```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>true_cluster</code>	Integer vector of true cluster assignments.
<code>K_true</code>	True number of clusters.
<code>r_true</code>	True factor dimension.
<code>select_result</code>	Result from select_K_r .
<code>runtime_sec</code>	Runtime in seconds (optional).

Value

A list with evaluation metrics:

`K_hat` Estimated number of clusters.
`r_hat` Estimated factor dimension.
`K_correct` Logical: `K_hat == K_true`.

r_correct Logical: $r_{\text{hat}} == r_{\text{true}}$.
 Kr_correct Logical: both K and r correct.
 ARI Adjusted Rand Index.
 VAF Variance Accounted For.
 logLik Log-likelihood.
 BIC Bayesian Information Criterion.
 runtime_sec Runtime in seconds.

Examples

```

## Not run:
# Generate data
data <- generate_synergy_data(N = 30, K = 3, r = 3, seed = 123)

# Run model selection
start_time <- Sys.time()
select_result <- select_K_r(data$list_of_data, method = "MFA", strategy = "grid")
runtime <- as.numeric(difftime(Sys.time(), start_time, units = "secs"))

# Evaluate
eval_result <- evaluate_model_selection(
  data$list_of_data, data$true_cluster,
  K_true = 3, r_true = 3,
  select_result, runtime_sec = runtime
)

## End(Not run)

```

extract_cluster_labels

Extract Cluster Labels from a Fitted Model

Description

This function extracts cluster assignment labels from a fitted MFA or MPCA model object. It searches through common field names used by different model implementations.

Usage

```
extract_cluster_labels(fit)
```

Arguments

fit A fitted model object (MFA or MPCA).

Details

The function searches for cluster labels in the following fields (in order):

- `z`: Standard field for MFA/MPCA models
- `cluster`: Alternative naming
- `clusters`: Plural form
- `class`: Classification result
- `z_hat`: Estimated assignments
- `cl`: Short form
- `hard_cluster`: Hard assignment from soft clustering
- `labels`: Generic label field

If the model has a `resp` (responsibility) matrix but no hard assignments, the function computes hard assignments using MAP (maximum a posteriori).

Value

An integer vector of cluster assignments (1 to K), or NA if cluster labels cannot be extracted.

Examples

```
## Not run:
fit <- mfa_em_fit(list_of_data, K = 3, r = 2)
pred_cluster <- extract_cluster_labels(fit)

## End(Not run)
```

extract_envelope

Extract Envelope from EMG Signal

Description

Extracts the amplitude envelope from rectified EMG signals using various methods.

Usage

```
extract_envelope(
  data,
  sampling_rate,
  method = "lowpass",
  cutoff = 6,
  window_ms = 50,
  filter_order = 4,
  zero_phase = TRUE
)
```



```
## End(Not run)
```

```
extract_selection_summary
```

Extract Selection Summary as Data Frame

Description

Converts the summary from `select_K_r` to a flat data frame, useful for saving and plotting.

Usage

```
extract_selection_summary(select_result)
```

Arguments

`select_result` Result from `select_K_r`.

Value

A data frame with columns K, r, logLik, BIC, and optionally stage (for twostage strategy).

```
extract_subject_features_by_singlePCA
```

Extract subject-level features by single PCA

Description

This function applies PCA to the concatenated data across all subjects, then for each subject i , computes the mean of the top r_dim principal component scores as a single feature vector. This is useful for initializing cluster assignments in mixture models.

Usage

```
extract_subject_features_by_singlePCA(list_of_data, r_dim = 2)
```

Arguments

`list_of_data` A list of $(T_i \times M)$ matrices, where each element represents time-series data for one subject. T_i is the number of time points for subject i , and M is the number of variables (e.g., muscles in EMG data).

`r_dim` Number of principal components to extract. Defaults to 2.

Details

The function first concatenates all subject data into a single matrix, performs PCA on this combined dataset, and then computes the mean of the PC scores for each subject's time points. This provides a low-dimensional representation of each subject that can be used for clustering initialization.

Value

A numeric matrix of shape (N x r_dim), where N=length(list_of_data). Each row contains the mean PC scores for one subject.

Examples

```
## Not run:
# Create example data
N <- 10
M <- 8
list_of_data <- lapply(1:N, function(i) matrix(rnorm(100 * M), 100, M))

# Extract 2D features for each subject
features <- extract_subject_features_by_singlePCA(list_of_data, r_dim = 2)
dim(features) # Should be 10 x 2

## End(Not run)
```

facet_mixsynergy

Apply Consistent Faceting for Cluster and Factor Visualizations

Description

Helper function to apply consistent facet_grid() styling for cluster-factor plots.

Usage

```
facet_mixsynergy(rows = NULL, cols = NULL, scales = "fixed", ...)
```

Arguments

rows	Variable(s) for facet rows (e.g., "Cluster")
cols	Variable(s) for facet columns (e.g., "Factor")
scales	Scale behavior: "fixed" (default), "free", "free_x", "free_y"
...	Additional arguments passed to facet_grid()

Value

A ggplot2 facet_grid object

Examples

```
## Not run:
library(ggplot2)
ggplot(data, aes(x, y)) +
  geom_line() +
  facet_mixsynergy(rows = "Cluster", cols = "Factor", scales = "free_y")

## End(Not run)
```

fit_single_factor_analysis

Fit a Single Factor Analysis (Ignoring Clusters)

Description

This function concatenates all subjects' data into one large matrix, then performs a naive factor analysis. It returns a simple BIC measure and the fitted loadings, mean, and diagonal Psi.

Usage

```
fit_single_factor_analysis(list_of_data, r = 2)
```

Arguments

`list_of_data` A list of ($T_i \times M$) matrices.
`r` The factor dimension.

Value

A list with elements (loadings, μ , diagPsi, logLik, BIC).

Examples

```
## Not run:
sim <- simulate_mixture_data(N = 50, K = 3, r = 2, M = 6, T_each = 100)
fa_res <- fit_single_factor_analysis(sim$list_of_data, r = 2)
print(fa_res$BIC)

## End(Not run)
```

fit_single_pca	<i>Fit a Single PCA (Ignoring Clusters)</i>
----------------	---

Description

Concatenates all data into one matrix, performs SVD, keeps r principal components, then computes a naive PPCA logLik and BIC.

Usage

```
fit_single_pca(list_of_data, r = 2)
```

Arguments

list_of_data	List of ($T_i \times M$) matrices.
r	Number of principal components.

Value

A list with (P, mu, sigma2, logLik, BIC).

Examples

```
## Not run:
sim <- simulate_mixture_data(N = 50, K = 3, r = 2, M = 6, T_each = 100)
pca_res <- fit_single_pca(sim$list_of_data, r = 2)
print(pca_res$BIC)

## End(Not run)
```

fit_two_step_baseline	<i>Two-step Baseline Algorithm for PCA/FA</i>
-----------------------	---

Description

This function implements a two-step baseline approach for comparison with mixture-based methods:

1. **Step 1: Individual Decomposition** - Apply PCA or FA to each subject's data separately to extract r basis vectors per subject.
2. **Step 1.5: Basis Alignment** - Align basis vectors across subjects using the Hungarian algorithm to ensure corresponding components are in the same order, improving clustering accuracy.
3. **Step 2: Clustering** - Normalize the basis vectors, concatenate them across all subjects, and apply k-means clustering. Assign each subject to a cluster based on majority vote of their basis vectors.

Usage

```
fit_two_step_baseline(
  data_list,
  r,
  K,
  method = c("PCA", "FA"),
  seed = 123,
  z_true = NULL,
  refine_assignments = FALSE,
  align_basis = TRUE
)
```

Arguments

<code>data_list</code>	A list of ($T_i \times M$) matrices, one per subject.
<code>r</code>	Number of components (factors or principal components) to extract per subject.
<code>K</code>	Number of clusters for k-means.
<code>method</code>	Either "PCA" or "FA" for the decomposition method.
<code>seed</code>	Random seed for reproducibility.
<code>z_true</code>	Optional true cluster labels for computing ARI. If NULL, ARI is set to NA.
<code>refine_assignments</code>	Logical. If TRUE, performs a single reassignment pass after fitting cluster-specific models in Step 4. Default is FALSE.
<code>align_basis</code>	Logical. If TRUE (default), applies Hungarian algorithm alignment in Step 1.5 to ensure corresponding components across subjects are in the same order before clustering. Set to FALSE to disable alignment.

Details

The algorithm addresses sign indeterminacy by normalizing each basis vector so that its maximum absolute value element is positive. Tie-breaking in majority vote is handled by computing the mean distance from a subject's basis vectors to each tied cluster centroid and selecting the closer one.

Step 1.5 (basis alignment) uses the Hungarian algorithm to align basis vectors across subjects. This ensures that corresponding components (e.g., the first component of each subject) represent similar patterns, which is crucial for accurate clustering. Without alignment, the arbitrary ordering of components from individual PCA/FA can lead to poor clustering performance.

Step 4 (re-optimization) fits cluster-specific models in standardized space to ensure consistency with Step 1. This improves reconstruction metrics (SSE/VAF) but does not change ARI unless `refine_assignments = TRUE`.

If `refine_assignments = TRUE`, a single reassignment pass is performed after fitting cluster-specific models. Each subject is reassigned to the cluster that minimizes reconstruction error. This can improve ARI while maintaining the two-step nature of the baseline (unlike iterative EM which would drift toward the mixture model).

For FA decomposition, the function uses `psych::fa()` if available. If the `psych` package is not installed or if FA fails for a subject, the function falls back to PCA with a warning.

Value

A list with:

`z_est` Integer vector of estimated cluster assignments (1..K).
`W_list` List of normalized basis matrices, one per subject.
`W_centroids` Matrix of cluster centroids in basis space.
`mu_list` List of centering vectors used for standardization, one per subject.
`sd_list` List of scaling vectors used for standardization, one per subject.
`cluster_models` List of cluster-specific models (each with `W` and `n_members`).
`method` Character string indicating the method used (e.g., "TwoStep_PCA").
`BIC` Set to NA (k-means does not have a canonical BIC).
`ARI` Adjusted Rand Index if `z_true` is provided, otherwise NA.

Examples

```
## Not run:
sim <- simulate_mixture_data(N = 50, K = 3, r = 2, M = 6, T_each = 100)
result <- fit_two_step_baseline(
  sim$list_of_data, r = 2, K = 3, method = "PCA",
  seed = 123, z_true = sim$z_true
)
print(result$ARI)

## End(Not run)
```

generate_pbs_script	<i>Generate PBS Job Script</i>
---------------------	--------------------------------

Description

Generates a PBS array job script for submitting parameter sweep tasks.

Usage

```
generate_pbs_script(
  n_tasks,
  output_dir,
  pbs_config = list(),
  sweep_type = c("modelsel", "baselines"),
  sweep_config = list()
)
```

Arguments

n_tasks	Integer. Total number of tasks (rows in parameter grid).
output_dir	Character. Directory for output files.
pbs_config	List. PBS configuration options: • job_name: Job name (default: "param_sweep") • queue: Queue name (default: NULL, use system default) • walltime: Wall time limit (default: "04:00:00") • ncpus: CPUs per task (default: 1) • mem: Memory per task (default: "4gb") • r_module: R module to load (default: NULL) • extra_modules: Additional modules to load (default: NULL) • r_path: Path to Rscript (default: "Rscript") • project: Project/account code (default: NULL) • extra_pbs_options: Additional PBS directives (default: NULL)
sweep_type	Character. Type of sweep: "modelsel" or "baselines".
sweep_config	List. Sweep-specific configuration passed to run_single_task.

Value

Character. Path to the generated PBS script.

Examples

```
## Not run:
script_path <- generate_pbs_script(
  n_tasks = 100,
  output_dir = "results",
  pbs_config = list(
    queue = "batch",
    walltime = "02:00:00",
    r_module = "R/4.3.0"
  )
)

## End(Not run)
```

generate_synergy_data *Generate v2 Dynamic Synergy Data for Model Selection Experiments*

Description

This function provides a standardized interface for generating v2 dynamic synergy data suitable for model selection experiments. It wraps [simulate_dynamic_synergy_data](#) and returns data in a format optimized for parameter sweep experiments with model selection.

Usage

```
generate_synergy_data(
  N = 30,
  K = 3,
  r = 3,
  M = 8,
  T_each = 200,
  cluster_sep_spatial = 1,
  cluster_sep_temporal = 1,
  cluster_sep_stability = 1,
  freq_range = c(0.5, 3),
  amp_range = c(0.5, 1.5),
  base_level = 0.05,
  rectify_output = TRUE,
  seed
)
```

Arguments

N	Number of subjects (time series). Default is 30.
K	Number of clusters (subgroups). Default is 3.
r	Number of synergies per cluster. Default is 3.
M	Number of observed channels (muscles). Default is 8.
T_each	Time-series length per subject. Default is 200.
cluster_sep_spatial	Numeric scalar (0-2) controlling spatial structure differentiation between clusters. Default is 1.0.
cluster_sep_temporal	Numeric scalar (0-2) controlling temporal coordination differentiation between clusters. Default is 1.0.
cluster_sep_stability	Numeric scalar (0-2) controlling stability-adaptability differentiation between clusters. Default is 1.0.
freq_range	Numeric vector of length 2 specifying the range of base frequencies (in Hz) for the periodic activation signals. Default is c(0.5, 3.0).
amp_range	Numeric vector of length 2 specifying the range of amplitudes for the periodic activation signals. Default is c(0.5, 1.5).
base_level	Numeric scalar specifying the baseline activation level. Default is 0.05.
rectify_output	Logical; if TRUE, apply half-wave rectification. Default is TRUE.
seed	Random seed for reproducibility. Required for deterministic results.

Details

This function is designed for use in parameter sweep experiments where model selection (K , r) is performed on generated data. The key differences from [simulate_dynamic_synergy_data](#) are:

- Returns `list_of_data` (renamed from `X_list`) for consistency with fitting functions
- Returns `true_cluster` (renamed from `z_true`) for consistency with evaluation functions
- Includes a `params` list for tracking generation parameters
- Seed is required (not optional) to ensure reproducibility

Value

A list with the following components:

`list_of_data` List of length `N`; each element is a (`T_each` x `M`) matrix of generated EMG signals.

This is the standard input format for MFA/MPCA fitting.

`true_cluster` Integer vector of length `N` indicating true cluster assignments (1 to `K`). Named as `true_cluster` for consistency.

`true_Lambda_list` List of length `K`; each element is a (`M` x `r`) matrix of true cluster-specific synergy loadings. Essential for subspace recovery evaluation. This is the canonical name for ground truth basis matrices.

`params` List containing all generation parameters for reproducibility. Includes both canonical short names (`T`, `sep_spatial`, `sep_temporal`, `sep_stability`) and full names (`T_each`, `cluster_sep_spatial`, etc.) for compatibility. Also includes `r_mode` = "global" for future extension.

`Lambda_list` List of length `K`; each element is a (`M` x `r`) matrix of cluster-specific synergy loadings. Kept for backward compatibility; identical to `true_Lambda_list`.

`C_list` List of length `N`; each element is a (`T_each` x `r`) matrix of synergy activation signals (for visualization/validation).

`spatial_params` List containing spatial structure parameters for each cluster.

`temporal_params` List containing temporal coordination parameters for each cluster.

`stability_params` List containing stability-adaptability parameters for each cluster.

See Also

[simulate_dynamic_synergy_data](#) for the underlying simulation function, [run_param_sweep_modelsel](#) for parameter sweep experiments.

Examples

```
## Not run:
# Generate data for model selection experiment
data <- generate_synergy_data(
  N = 30, K = 3, r = 3, M = 8, T_each = 200,
  cluster_sep_spatial = 1.0,
  cluster_sep_temporal = 1.0,
  cluster_sep_stability = 1.0,
  seed = 123
)

# Access the data for fitting
list_of_data <- data$list_of_data
true_cluster <- data$true_cluster
```

```
# Fit MFA model
fit <- mfa_em_fit(list_of_data, K = 3, r = 3)

# Evaluate clustering accuracy
ari <- mclust::adjustedRandIndex(true_cluster, fit$z)

## End(Not run)
```

get_model_by_K_r	<i>Retrieve a Specific Model by (K, r) from a Grid Search</i>
------------------	---

Description

After calling [select_optimal_K_r_mfa](#) (or a similar function), you have `$all_models` containing fits for various (K, r). This function extracts the model for a user-specified pair (K_target, r_target).

Usage

```
get_model_by_K_r(selection_obj, K_target, r_target)
```

Arguments

selection_obj	The object returned by select_optimal_K_r_mfa .
K_target	Desired number of clusters.
r_target	Desired factor dimension.

Value

The fitted model (the same structure returned by `mfa_em_fit`), or NULL if no match found.

Examples

```
## Not run:
sel_obj <- select_optimal_K_r_mfa(my_data, Kvec=2:4, rvec=1:3)
best_model <- get_model_by_K_r(sel_obj, K_target=3, r_target=2)

## End(Not run)
```

`get_model_by_K_r_mpca` *Retrieve a Mixture PCA Model by (K, r)*

Description

Given a `selection_obj` from `select_optimal_K_r_mpca`, extracts the model for the specified (`K_target`, `r_target`) if it exists.

Usage

```
get_model_by_K_r_mpca(selection_obj, K_target, r_target)
```

Arguments

<code>selection_obj</code>	The output of <code>select_optimal_K_r_mpca</code> .
<code>K_target</code>	Desired K.
<code>r_target</code>	Desired r.

Value

The fitted model (same structure as from `mixture_pca_em_fit`), or NULL.

`make_table1` *Create a Static Table of Simulation Conditions (Table 1)*

Description

This function returns a small `data.frame` that describes the parameter ranges used in the simulation (e.g., K, r, N, M, T_each, etc.).

Usage

```
make_table1()
```

Value

A `data.frame` with columns `Parameter` and `Values`.

Examples

```
## Not run:
table1 <- make_table1()
print(table1)

## End(Not run)
```

make_table2

Summarize Results for a Subset (Table 2)

Description

This function filters the data for certain separation/noise conditions (e.g., `sep=1.0` and `noise=1.0`) and computes group-wise means and standard deviations for ARI and BIC. It returns a small summary table.

Usage

```
make_table2(df)
```

Arguments

`df` A `data.frame` with columns like `ARI_MFA`, `ARI_PCA`, `BIC_MFA`, `BIC_PCA`.

Value

A `data.frame` summarizing ARI/BIC means and sds, grouped by (`K_true`, `r_true`, `N`, `M`).

Examples

```
## Not run:
df <- read_all_results("results")
table2 <- make_table2(df)
print(table2)

## End(Not run)
```

mfa_em_fit

Fit a Mixture Factor Analysis Model (with optional multi-initialization)

Description

This function calls the C++ function `mfaTimeseriesCpp()` to perform a Mixture Factor Analysis EM algorithm for fixed `K` and `r`. By default, it runs a single pass with the internal initialization from C++. However, if you specify multiple initial attempts (via `n_init>1`) and/or `use_kmeans_init=TRUE`, this function will try several different initial cluster assignments (in parallel using `mclapply`), then return the best solution (maximizing the final log-likelihood).

Usage

```

mfa_em_fit(
  list_of_data,
  K,
  r,
  max_iter = 50,
  iter_fa = 20,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  kmeans_rdim = r,
  mc_cores = 1,
  seed = NULL,
  verbose = TRUE,
  n_threads = 0,
  cpp_seed = 0,
  progress_callback = NULL,
  ...
)

```

Arguments

<code>list_of_data</code>	A list of matrices (each $(T_i \times M)$) to be modeled.
<code>K</code>	Number of clusters.
<code>r</code>	Factor dimension.
<code>max_iter</code>	Maximum EM iterations for <code>mfaTimeseriesCpp</code> .
<code>iter_fa</code>	Number of sub-iterations for the factor analyzer update in C++. (Replaces deprecated <code>nIterFA</code>)
<code>tol</code>	Convergence tolerance for <code>mfaTimeseriesCpp</code> .
<code>n_init</code>	Number of random initial assignments to try (in addition to the default single-run or k-means if requested). Defaults to 1.
<code>use_kmeans_init</code>	If TRUE, we also run one initialization where we extract subject features (via PCA) and apply kmeans to get a cluster assignment. Defaults to FALSE.
<code>kmeans_rdim</code>	PCA dimension for extracting subject-level features, used only if <code>use_kmeans_init=TRUE</code> . Default is <code>r</code> . (Replaces deprecated <code>subject_rdim_for_kmeans</code>)
<code>mc_cores</code>	Number of cores for parallel execution via <code>mclapply</code> . Defaults to 1 (no parallel).
<code>seed</code>	Random seed for reproducibility (default: NULL).
<code>verbose</code>	Logical; if TRUE, print progress messages during EM iterations. Default is TRUE.
<code>n_threads</code>	Integer; number of OpenMP threads to use for parallel computation (0 = auto). Default is 0.
<code>cpp_seed</code>	Integer; random seed for C++ RNG (0 = default/no seed). Default is 0.
<code>progress_callback</code>	Optional function to report progress from C++. Should accept two arguments: current iteration and total iterations.

... Additional arguments for backwards compatibility. Deprecated arguments like `nIterFA` and `subject_rdim_for_kmeans` are accepted with warnings.

Details

The single-run approach in C++ already does a built-in initialization if `z_init` is not provided. Therefore, if `n_init=1` and `use_kmeans_init=FALSE`, we just call `mfaTimeseriesCpp` once (like the original design). Otherwise:

1. If `use_kmeans_init=TRUE`, we do one run where we assign clusters by k-means on some subject-level features (extracted by PCA).
2. We also sample `n_init` random assignments (each subject assigned to a random cluster).
3. For each assignment, we call `mfa_em_fit_cpp_singleInit`.
4. We compare all results' `logLik` and keep the best one.

This provides a more robust initialization scheme to avoid poor local maxima.

Value

A list with the same elements as a single run: `z`, `pi`, `mu`, `Lambda`, `Psi`, `logLik`, `resp`. It corresponds to the best solution (i.e. highest log-likelihood) among all tried inits.

See Also

`mixture_pca_em_fit` for MPCA fitting, `select_r` for selecting optimal `r`, `plot_cluster_synergy_loadings_mfa` for visualization

Examples

```
## Not run:
# Suppose we have a list_of_data for N=100 subjects, each (T_i x M=8).
# We want to fit K=3, r=2, and try 5 random inits plus one k-means init:
fit <- mfa_em_fit(
  list_of_data, K=3, r=2,
  max_iter=50, iter_fa=20, tol=1e-3,
  n_init=5,
  use_kmeans_init=TRUE,
  kmeans_rdim=2,
  mc_cores=2,
  seed=123
)
print(fit$logLik)
head(fit$z)

## End(Not run)
```

mfa_em_fit_cpp_singleInit

Single-Run Mixture Factor Analysis with a Preset Initial Assignment

Description

This helper function calls the C++ routine `mfaTimeseriesCpp()` with a user-specified cluster assignment (`z_init`) and then computes the final log-likelihood and responsibilities in R.

Usage

```
mfa_em_fit_cpp_singleInit(
  list_of_data,
  K,
  r,
  z_init,
  max_iter = 50,
  nIterFA = 20,
  tol = 0.001,
  verbose = TRUE,
  n_threads = 0,
  seed = 0,
  progress_callback = NULL,
  em_type = c("hard", "soft")
)
```

Arguments

<code>list_of_data</code>	A list of ($T_i \times M$) matrices, one for each subject.
<code>K</code>	Number of clusters.
<code>r</code>	Factor dimension in each cluster.
<code>z_init</code>	An integer vector of length N (where $N = \text{length}(\text{list_of_data})$), giving the initial cluster label ($1..K$) for each subject.
<code>max_iter</code>	Maximum EM iterations in C++.
<code>nIterFA</code>	Number of sub-iterations for the factor analyzer update in C++.
<code>tol</code>	Convergence tolerance used in C++.
<code>verbose</code>	Logical; if TRUE, print progress messages. Default is TRUE.
<code>n_threads</code>	Integer; number of OpenMP threads to use (0 = auto). Default is 0.
<code>seed</code>	Integer; random seed for C++ RNG (0 = default/no seed). Default is 0.
<code>progress_callback</code>	Optional function to report progress from C++. Should accept two arguments: current iteration and total iterations.
<code>em_type</code>	Character; type of EM algorithm to use. Either "hard" (classification EM with hard assignments) or "soft" (standard EM with responsibility-weighted updates). Default is "hard".

Value

A list with elements:

`z` Hard cluster assignments, length N .
`pi` Cluster mixing proportions, length K .
`mu` List of length K , each a mean vector ($M \times 1$).
`Lambda` List of length K , each an $M \times r$ factor loading matrix.
`Psi` List of length K , each an $M \times M$ diagonal noise matrix.
`logLik` Final log-likelihood computed in R.
`resp` An $N \times K$ matrix of responsibilities.

Examples

```
## Not run:
# Suppose we have list_of_data, K=3, r=2:
z_init <- sample.int(3, size=length(list_of_data), replace=TRUE)
fit_one <- mfa_em_fit_cpp_singleInit(list_of_data, K=3, r=2,
                                   z_init=z_init)

fit_one$logLik

## End(Not run)
```

mixture_pca_em_fit	<i>Fit a Mixture PCA Model (Using mpcaTimeseriesCpp) with optional multi-initialization</i>
--------------------	---

Description

This function calls the C++ function `mpcaTimeseriesCpp()` to perform a Mixture PCA (PPCA) EM algorithm for fixed K and r , then converts the W matrices into (P, D) , creates Ψ , and computes the final log-likelihood/responsibilities in R. By default, it runs a single pass with the built-in initialization in C++. However, if `n_init > 1` or `use_kmeans_init=TRUE`, it performs multiple initializations in parallel and returns the best-fitting result (highest log-likelihood).

Usage

```
mixture_pca_em_fit(
  list_of_data,
  K,
  r,
  max_iter = 50,
  iter_pca = 20,
  tol = 0.001,
  method = "EM",
  n_init = 1,
```

```

    use_kmeans_init = TRUE,
    kmeans_rdim = r,
    mc_cores = 1,
    seed = NULL,
    verbose = TRUE,
    n_threads = 0,
    cpp_seed = 0,
    progress_callback = NULL,
    ...
)

```

Arguments

<code>list_of_data</code>	A list of matrices (each $T_i \times M$).
<code>K</code>	Number of clusters.
<code>r</code>	Number of principal components.
<code>max_iter</code>	Maximum EM iterations for the C++ routine.
<code>iter_pca</code>	Sub-iterations for updating each cluster's PPCA parameters. (Replaces deprecated <code>nIterPCA</code>)
<code>tol</code>	Convergence tolerance for <code>mpcaTimeseriesCpp</code> .
<code>method</code>	Either "EM" or "closed_form", passed down to the C++ routine.
<code>n_init</code>	Integer; how many random initial assignments to try (in addition to the default single-run or k-means if requested). Defaults to 1.
<code>use_kmeans_init</code>	Logical; if TRUE, we also run one initialization where we assign clusters by k-means on subject-level PCA features. Defaults to FALSE.
<code>kmeans_rdim</code>	The PCA dimension for the subject-level feature extraction, used only if <code>use_kmeans_init=TRUE</code> . Defaults to <code>r</code> . (Replaces deprecated <code>subject_rdim_for_kmeans</code>)
<code>mc_cores</code>	Number of cores for parallel execution via <code>mclapply</code> . Defaults to 1 (no parallel).
<code>seed</code>	Random seed for reproducibility (default: NULL).
<code>verbose</code>	Logical; if TRUE, print progress messages during EM iterations. Default is TRUE.
<code>n_threads</code>	Integer; number of OpenMP threads to use for parallel computation (0 = auto). Default is 0.
<code>cpp_seed</code>	Integer; random seed for C++ RNG (0 = default/no seed). Default is 0.
<code>progress_callback</code>	Optional function to report progress from C++. Should accept two arguments: current iteration and total iterations.
<code>...</code>	Additional arguments for backwards compatibility. Deprecated arguments like <code>nIterPCA</code> and <code>subject_rdim_for_kmeans</code> are accepted with warnings.

Details

If `n_init=1` and `use_kmeans_init=FALSE`, this function runs exactly one pass with the default initialization in C++ (i.e. subject `i` is assigned to cluster $(i \% K) + 1$). Otherwise:

1. If use_kmeans_init=TRUE, we do a run where we apply k-means to some subject-level PCA features to get z_init, then run EM once.
2. We generate n_init random initial assignments, run EM for each in parallel, and collect the results.
3. We compare all solutions by final log-likelihood and pick the best.

Value

A list with elements:

z	Hard cluster assignments (1..K).
pi	Mixing proportions.
mu	List of length K, each a mean vector.
P	List of length K, each ($M \times r$) principal directions.
D	List of length K, each ($r \times r$) diagonal.
Psi	List of length K, each ($M \times M$) diagonal.
logLik	Final log-likelihood over all data.
sigma2	Numeric vector of length K for sigma2.
resp	($N \times K$) matrix of responsibilities.

See Also

[mfa_em_fit](#) for MFA fitting, [select_r](#) for selecting optimal r, [plot_cluster_synergy_loadings_mPCA](#) for visualization

Examples

```
## Not run:
# Suppose we have a list_of_data, K=3, r=2, and we want to try 5 random inits + k-means:
fit <- mixture_pca_em_fit(
  list_of_data, K=3, r=2, max_iter=50, iter_pca=20, tol=1e-3, method="EM",
  n_init=5, use_kmeans_init=TRUE, kmeans_rdim=2, mc_cores=2
)
print(fit$logLik)
head(fit$z)

## End(Not run)
```

mixture_pca_em_fit_cpp_singleInit

Single-Run Mixture PCA with a Preset Initial Assignment

Description

This helper function calls the updated C++ routine `mpcaTimeseriesCpp()` with a user-specified cluster assignment (`z_init`), then converts the returned `W` into (`P`, `D`) and constructs `Psi`, and finally computes the final log-likelihood and responsibilities in `R`.

Usage

```
mixture_pca_em_fit_cpp_singleInit(
  list_of_data,
  K,
  r,
  max_iter = 50,
  nIterPCA = 20,
  tol = 0.001,
  method = "EM",
  z_init,
  verbose = TRUE,
  n_threads = 0,
  seed = 0,
  progress_callback = NULL,
  em_type = c("hard", "soft")
)
```

Arguments

<code>list_of_data</code>	A list of matrices (each $T_i \times M$).
<code>K</code>	Number of clusters.
<code>r</code>	Number of principal components (dimension).
<code>max_iter</code>	Maximum EM iterations (C++).
<code>nIterPCA</code>	Sub-iterations for updating each cluster's PPCA parameters in C++.
<code>tol</code>	Convergence tolerance (C++).
<code>method</code>	Either "EM" or "closed_form", passed to <code>mpcaTimeseriesCpp</code> .
<code>z_init</code>	An integer vector of length N giving initial cluster labels (1..K).
<code>verbose</code>	Logical; if TRUE, print progress messages. Default is TRUE.
<code>n_threads</code>	Integer; number of OpenMP threads to use (0 = auto). Default is 0.
<code>seed</code>	Integer; random seed for C++ RNG (0 = default/no seed). Default is 0.
<code>progress_callback</code>	Optional function to report progress from C++. Should accept two arguments: current iteration and total iterations.

em_type Character; type of EM algorithm to use. Either "hard" (classification EM with hard assignments) or "soft" (standard EM with responsibility-weighted updates). Default is "hard".

Value

A list with elements:

z	Hard cluster assignments (1..K).
pi	Mixing proportions (length K).
mu	List of length K, each a mean vector (length M).
W	List of length K, each an (M x r) loading matrix (optional).
P	List of length K, each an (M x r) matrix of principal directions.
D	List of length K, each an (r x r) diagonal matrix of column norms.
Psi	List of length K, each an (M x M) diagonal matrix $\sigma^2 \cdot I$.
logLik	Final log-likelihood computed in R.
sigma2	Numeric vector of length K, the final σ^2 for each cluster.
resp	An (N x K) matrix of responsibilities.

Examples

```
## Not run:
# Suppose we have list_of_data, K=3, r=2, method="EM", plus some z_init:
z_init <- sample.int(3, size=length(list_of_data), replace=TRUE)
fit_one <- mixture_pca_em_fit_cpp_singleInit(
  list_of_data, K=3, r=2, max_iter=50, nIterPCA=20, tol=1e-3,
  method="EM", z_init=z_init
)
print(fit_one$logLik)
# Now we can call compute_factor_scores_mPCA(fit_one, list_of_data) because fit_one$W exists

## End(Not run)
```

normalize_emg

Normalize EMG Signal

Description

Normalizes EMG signals using various methods to allow comparison across channels, subjects, or sessions.

Usage

```
normalize_emg(data, method = "max", reference = NULL, per_channel = TRUE)
```

Arguments

data	EMG signal. Can be a numeric vector (single channel), a matrix (T x M, time x channels), or a list of matrices.
method	Normalization method: • "max": Divide by maximum value (default) • "mvc": Divide by Maximum Voluntary Contraction reference • "mean": Divide by mean value • "zscore": Z-score standardization (subtract mean, divide by SD) • "range": Min-max normalization to [0, 1] • "median": Divide by median value (robust to outliers)
reference	Reference values for MVC normalization. Can be a single value (applied to all channels) or a vector (one value per channel). Only used when method = "mvc".
per_channel	Logical. If TRUE (default), normalize each channel independently. If FALSE, use global statistics across all channels.

Details

Normalization is essential for:

- Comparing EMG across different muscles
- Comparing EMG across different subjects
- Comparing EMG across different recording sessions

MVC normalization (method = "mvc") is the gold standard in EMG research, expressing activation as a percentage of maximum voluntary contraction. Requires separate MVC trials for reference.

Max normalization (method = "max") is useful when MVC data is not available, normalizing to the peak value in the data.

Value

Normalized EMG signal, same structure as input.

Examples

```
# Max normalization (0 to 1 range)
emg <- matrix(rnorm(1000), ncol = 4)
norm_emg <- normalize_emg(emg, method = "max")

# MVC normalization (% MVC)
mvc_values <- c(100, 150, 80, 120) # MVC for each channel
norm_emg <- normalize_emg(emg, method = "mvc", reference = mvc_values)

# Z-score normalization
norm_emg <- normalize_emg(emg, method = "zscore")
```

notch_filter

Notch Filter for Power Line Noise Removal

Description

Applies a notch (band-stop) filter to remove power line interference (50 Hz or 60 Hz) and optionally its harmonics.

Usage

```
notch_filter(  
    data,  
    sampling_rate,  
    notch_freq = 50,  
    Q = 30,  
    harmonics = TRUE,  
    zero_phase = TRUE  
)
```

Arguments

data	EMG signal. Can be a numeric vector (single channel), a matrix (T x M, time x channels), or a list of matrices.
sampling_rate	Sampling frequency in Hz.
notch_freq	Frequency to remove in Hz. Default is 50 Hz. Use 60 Hz for regions with 60 Hz power lines (e.g., Americas, Japan).
Q	Quality factor determining the filter bandwidth. Higher Q = narrower notch. Default is 30.
harmonics	Logical. If TRUE (default), also removes harmonics (2x, 3x, etc. of notch_freq up to Nyquist frequency).
zero_phase	Logical. If TRUE (default), applies zero-phase filtering.

Details

Power line interference (50/60 Hz) is a common artifact in EMG recordings. The notch filter creates a narrow band-stop at the specified frequency. When `harmonics = TRUE`, harmonics (100/120 Hz, 150/180 Hz, etc.) are also removed.

Value

Filtered EMG signal with power line noise removed.

Examples

```
## Not run:
# Remove 50 Hz power line noise
clean_emg <- notch_filter(emg, sampling_rate = 1000, notch_freq = 50)

# Remove 60 Hz (Americas) with harmonics
clean_emg <- notch_filter(emg, sampling_rate = 1000, notch_freq = 60, harmonics = TRUE)

## End(Not run)
```

plot.mixture_r_selection

Plot Method for mixture_r_selection Objects

Description

Visualize r-selection results with multiple plot types.

Usage

```
## S3 method for class 'mixture_r_selection'
plot(
  x,
  type = c("overview", "global_cv", "cluster_metrics", "r_comparison", "cluster_scores"),
  ...
)
```

Arguments

x	A mixture_r_selection object.
type	Type of plot to generate. Options: "overview" (default): Multi-panel overview showing all available plots "global_cv": Global CV curve (only for hybrid/global-first modes with CV criterion) "cluster_metrics": Heatmap of cluster-specific metrics across r values "r_comparison": Bar plot comparing r_global and cluster-specific r values "cluster_scores": Line plot of composite scores for each cluster across r values
...	Additional arguments (ignored).

Details

The plot method provides several visualization types depending on the selection mode:

For global-first mode: - "global_cv": Shows CV error curve with selected `r_global`

For clusterwise mode: - "cluster_metrics": Heatmap of VAF/VSS/smoothness for each cluster and `r` value - "cluster_scores": Composite scores showing how each cluster's metrics vary with `r`

For hybrid mode: - All of the above plus "r_comparison" showing `r_global` vs cluster-specific `r` values

The "overview" type automatically generates a multi-panel plot with all relevant visualizations for the given mode.

Examples

```
## Not run:
result <- select_r_mixture(list_of_data, K = 3, mode = "hybrid")
plot(result) # Overview
plot(result, type = "global_cv")
plot(result, type = "cluster_metrics")
plot(result, type = "r_comparison")

## End(Not run)
```

plot_activation_patterns

Plot Activation Patterns

Description

Visualizes temporal activation curves per cluster by showing representative activation patterns from subjects in each cluster.

Usage

```
plot_activation_patterns(sim, n_subjects_per_cluster = 3, theme_base_size = 14)
```

Arguments

<code>sim</code>	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>C_list</code> and <code>z_true</code> components.
<code>n_subjects_per_cluster</code>	Number of subjects to display per cluster. Default is 3.
<code>theme_base_size</code>	Base font size for the plot theme. Default is 14.

Value

A ggplot object showing temporal activation curves per cluster.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_activation_patterns(sim)

## End(Not run)
```

```
plot_aligned_basis_heatmaps
```

Plot Aligned Basis Heatmaps

Description

Creates heatmaps of basis vectors (W matrices) for each subject and component, showing how well the Two-step alignment procedure worked. Compares aligned basis vectors with true synergies (if available) to diagnose alignment failures.

Usage

```
plot_aligned_basis_heatmaps(
  W_list,
  W_true = NULL,
  z_true = NULL,
  z_est = NULL,
  max_subjects = 30,
  theme_base_size = 10
)
```

Arguments

<code>W_list</code>	List of basis matrices, one per subject. Each matrix should be $M \times r$ (M channels/muscles, r components/synergies). This is typically the output from <code>fit_two_step_baseline()</code> $\$W_list$.
<code>W_true</code>	Optional list of true basis matrices for comparison (from simulation data, e.g., <code>sim\$Lambda_list</code>). If provided, true synergies will be shown alongside estimated ones. Default is <code>NULL</code> .
<code>z_true</code>	Optional vector of true cluster assignments (length N). Used to organize subjects by cluster. If <code>NULL</code> , subjects are shown in original order.
<code>z_est</code>	Optional vector of estimated cluster assignments (length N). Used to show clustering results alongside basis vectors. Default is <code>NULL</code> .
<code>max_subjects</code>	Maximum number of subjects to display. If $N > \text{max_subjects}$, a random sample will be shown. Default is 30.
<code>theme_base_size</code>	Base font size for the plot theme. Default is 10.

Details

This function visualizes the output of the Two-step alignment procedure (`align_basis_across_subjects()`) to diagnose alignment failures. Each row represents one subject's basis vector for a specific component. Columns represent muscles/channels. Colors indicate the weight/loading values.

When `W_true` is provided, the function creates a side-by-side comparison showing true synergies (left panel) and estimated aligned basis vectors (right panel). This helps identify whether alignment failures are due to:

- Poor individual decomposition (Step 1)
- Alignment algorithm issues (Step 1.5)
- Fundamental differences between true and estimated structures

Value

A ggplot object showing heatmaps of basis vectors organized by subject and component, with optional comparison to true synergies.

Examples

```
## Not run:
# Simulate data with known ground truth
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)

# Fit Two-step baseline
fit <- fit_two_step_baseline(
  sim$list_of_data, r=3, K=3, method="PCA",
  z_true=sim$z_true, align_basis=TRUE
)

# Visualize aligned basis vectors
plot_aligned_basis_heatmaps(fit$W_list, z_est=fit$z_est)

# Compare with true synergies
plot_aligned_basis_heatmaps(
  fit$W_list,
  W_true=sim$Lambda_list,
  z_true=sim$z_true,
  z_est=fit$z_est
)

## End(Not run)
```

plot_all_factor_scores_mfa

Plot All Factor Scores (Z) for MFA, Faceted by Cluster and Factor

Description

This function computes the factor scores $Z_i(t)$ for each subject i , identifies that subject's cluster k , and then creates a large facet plot of the time evolution of each factor (1.. r) in each cluster (1.. K). Optionally overlays all subjects in the same cluster.

Usage

```
plot_all_factor_scores_mfa(
  mfa_fit,
  list_of_data,
  overlay_subjects = TRUE,
  muscle_names = NULL
)
```

Arguments

mfa_fit	A fitted MFA model (with z , Λ , Ψ , μ , etc.).
list_of_data	A list of ($T_i \times M$) data matrices.
overlay_subjects	Logical; if TRUE, all subjects in the same cluster are overlaid in the same facet. If FALSE, you could average them, etc.
muscle_names	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object that shows the factor scores over time, with facets by (Cluster, Factor).

Examples

```
## Not run:
p <- plot_all_factor_scores_mfa(my_mfa_fit, list_of_data)
print(p)

## End(Not run)
```

plot_all_factor_scores_mPCA

Plot All Factor Scores (Z) for MPCA, Faceted by Cluster and Factor

Description

This function uses precomputed factor scores `mpca_fit$factor_scores[[i]]` (each $T_i \times r$) and plots them over time, grouped by cluster $z[i]$. Optionally overlays all subjects in the same cluster on one facet, or can average them.

Usage

```
plot_all_factor_scores_mpca(
  mpca_fit,
  list_of_data,
  overlay_subjects = TRUE,
  muscle_names = NULL
)
```

Arguments

mpca_fit	A fitted MPCA model (with z , W , μ , etc.) plus factor_scores[[i]] if available.
list_of_data	A list of ($T_i \times M$) data matrices (just for dimension reference).
overlay_subjects	Logical; if TRUE, all subjects in the same cluster are overlaid in the same facet. If FALSE, we can average them etc.
muscle_names	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object that shows the factor scores over time, with facets by (Cluster, Factor).

Examples

```
## Not run:
mpca_fit <- compute_factor_scores_mpca(mpca_fit, list_of_data)
p <- plot_all_factor_scores_mpca(mpca_fit, list_of_data, overlay_subjects=TRUE)
print(p)

## End(Not run)
```

plot_all_reconstructed_waveforms_mfa

Plot Reconstructed Waveforms for All Clusters and Channels (MFA)

Description

This function plots the reconstructed waveforms ($\hat{X}_i$) for each subject across all channels, grouped by cluster. By default, it overlays all subjects in the same cluster, then uses facet to separate by (Cluster, Channel).

Usage

```
plot_all_reconstructed_waveforms_mfa(
  mfa_fit,
  list_of_data,
  overlay_subjects = TRUE,
  muscle_names = NULL
)
```

Arguments

mfa_fit	A fitted MFA model (with z, Lambda, Psi, mu, etc.).
list_of_data	A list of ($T_i \times M$) data matrices used in fitting.
overlay_subjects	Logical; if TRUE, each subject in the same cluster is drawn in the same facet. If FALSE, the function computes the average waveforms across subjects in each cluster.
muscle_names	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object with line plots of reconstructed waveforms, facet by cluster & channel.

Examples

```
## Not run:
p_recon <- plot_all_reconstructed_waveforms_mfa(mfa_fit, list_of_data)
print(p_recon)

## End(Not run)
```

```
plot_all_reconstructed_waveforms_mPCA
```

Plot Reconstructed Waveforms for All Clusters and Channels (MPCA)

Description

This function plots the reconstructed waveforms ($\hat{X}_i$) for each subject across all channels, grouped by cluster. By default, it overlays all subjects in the same cluster, then uses facet to separate by (Cluster, Channel).

Usage

```
plot_all_reconstructed_waveforms_mpca(
  mpca_fit,
  list_of_data,
  overlay_subjects = TRUE,
  muscle_names = NULL
)
```

Arguments

<code>mpca_fit</code>	A fitted MPCA model (with z , W , μ , etc.). Also <code>factor_scores[[i]]</code> must be available if we want to reconstruct $\hat{X}$.
<code>list_of_data</code>	A list of ($T_i \times M$) data matrices used in fitting.
<code>overlay_subjects</code>	Logical; if TRUE, each subject in the same cluster is drawn in the same facet. If FALSE, the function computes the average waveforms across subjects in each cluster.
<code>muscle_names</code>	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object with line plots of reconstructed waveforms, facet by cluster & channel.

Examples

```
## Not run:
mpca_fit <- compute_factor_scores_mpca(mpca_fit, list_of_data)
p_recon <- plot_all_reconstructed_waveforms_mpca(mpca_fit, list_of_data)
print(p_recon)

## End(Not run)
```

<code>plot_basis_umap</code>	<i>Plot Basis UMAP Embedding</i>
------------------------------	----------------------------------

Description

Creates a UMAP or PCA embedding of basis vectors at the component level (not subject level) to visualize clustering quality. Each point represents one basis vector (subject $\times$ component), colored by true cluster and shaped by estimated cluster. This helps diagnose clustering failures in the Two-step algorithm.

Usage

```
plot_basis_umap(
  W_list,
  z_true,
  z_est = NULL,
  method = "umap",
  theme_base_size = 14
)
```

Arguments

<code>W_list</code>	List of basis matrices, one per subject. Each matrix should be $M \times r$ (M channels/muscles, r components/synergies). This is typically the output from <code>fit_two_step_baseline()</code> <code>\$W_</code>
<code>z_true</code>	Vector of true cluster assignments (length N). Required for coloring points by true cluster.
<code>z_est</code>	Optional vector of estimated cluster assignments (length N). If provided, points will be shaped by estimated cluster to show mismatches. Default is <code>NULL</code> .
<code>method</code>	Character string specifying the embedding method: "umap" or "pca". Default is "umap".
<code>theme_base_size</code>	Base font size for the plot theme. Default is 14.

Details

This function extends `plot_cluster_embedding()` to work at the basis vector level rather than the subject level. Instead of showing N points (one per subject), it shows $N \times r$ points (one per basis vector).

This visualization is particularly useful for diagnosing Two-step clustering failures because it reveals:

- Whether basis vectors from the same true cluster are close together
- Whether k-means clustering in Step 2 correctly separated clusters
- Which specific basis vectors were misclassified
- Whether alignment in Step 1.5 successfully matched corresponding components

When `z_est` is provided, misclassified basis vectors are easy to spot as points with different colors (true cluster) and shapes (estimated cluster).

Value

A ggplot object showing low-dimensional embedding of basis vectors ($N \times r$ points), colored by true cluster and optionally shaped by estimated cluster.

Examples

```
## Not run:
# Simulate data with known ground truth
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)

# Fit Two-step baseline
fit <- fit_two_step_baseline(
  sim$list_of_data, r=3, K=3, method="PCA",
  z_true=sim$z_true, align_basis=TRUE
)

# Visualize basis vector embedding
plot_basis_umap(fit$W_list, z_true=sim$z_true, z_est=fit$z_est)

# Use PCA instead of UMAP
plot_basis_umap(fit$W_list, z_true=sim$z_true, z_est=fit$z_est, method="pca")

## End(Not run)
```

plot_bic_comparison *Plot BIC Comparison for Parameter Sweep*

Description

Creates a boxplot or violin plot comparing BIC values across methods and model types. Lower BIC values indicate better model fit.

Usage

```
plot_bic_comparison(aggregated_df, plot_type = "boxplot")
```

Arguments

aggregated_df	Data frame with aggregated results from run_param_sweep(). Must contain columns: Method, model_type, BIC_mean.
plot_type	Character. Type of plot. One of "boxplot" or "violin". Default is "boxplot".

Value

A ggplot2 object

Examples

```
## Not run:
library(synergyMixR)
results <- run_param_sweep(test = TRUE, cores = 2)

# Plot BIC comparison
```

```
p <- plot_bic_comparison(results$aggregated_results)
print(p)

## End(Not run)
```

plot_bic_line	<i>Line Plot of BIC vs. K or r</i>
---------------	------------------------------------

Description

Takes the summary data.frame from `select_optimal_K_r_mfa()` (which has columns K, r, logLik, BIC) and produces a simple line plot of BIC. You can choose whether the x-axis is K or r, and color/line by the other.

Usage

```
plot_bic_line(df_summary, x_axis = c("K", "r"))
```

Arguments

df_summary	A data.frame with columns K, r, BIC.
x_axis	One of "K" or "r". That variable is placed on the x-axis.

Value

A ggplot object.

plot_calibration	<i>Plot Calibration Curve</i>
------------------	-------------------------------

Description

Creates a reliability diagram (calibration plot) showing the relationship between predicted probabilities and observed frequencies.

Usage

```
plot_calibration(
  calib_result,
  show_histogram = TRUE,
  by_cluster = FALSE,
  title = "Calibration Plot",
  ...
)
```

Arguments

calib_result	Result from compute_calibration .
show_histogram	Logical. If TRUE, show histogram of predictions below the calibration curve. Default: TRUE.
by_cluster	Logical. If TRUE, show separate curves for each cluster. If FALSE, aggregate across clusters. Default: FALSE.
title	Character. Plot title. Default: "Calibration Plot".
...	Additional arguments passed to ggplot2 functions.

Details

A reliability diagram plots mean predicted probability (x-axis) against mean observed frequency (y-axis) for each bin. A perfectly calibrated model follows the diagonal line ($y = x$). Points above the diagonal indicate underconfidence; points below indicate overconfidence.

The shaded band around the diagonal shows +/- 0.1 deviation as a reference. The ECE value is displayed in the plot subtitle.

Value

A ggplot2 object.

See Also

[compute_calibration](#)

Examples

```
## Not run:
calib <- compute_calibration(fit$resp, z_true)

# Basic calibration plot
p <- plot_calibration(calib)
print(p)

# Show by cluster
p <- plot_calibration(calib, by_cluster = TRUE)
print(p)

# Save to file
ggsave("calibration.pdf", p, width = 6, height = 5)

## End(Not run)
```

`plot_cluster_embedding`*Plot Cluster Embedding*

Description

Creates a low-dimensional embedding (UMAP or PCA) to visualize cluster separability. Each point represents a subject, colored by true cluster assignment.

Usage

```
plot_cluster_embedding(sim, fit = NULL, method = "umap", theme_base_size = 14)
```

Arguments

<code>sim</code>	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>X_list</code> and <code>z_true</code> components.
<code>fit</code>	Optional fitted model object (e.g., from MFA/MPCA). If provided, predicted cluster assignments can be overlaid. Default is <code>NULL</code> .
<code>method</code>	Character string specifying the embedding method: "umap" or "pca". Default is "umap".
<code>theme_base_size</code>	Base font size for the plot theme. Default is 14.

Value

A ggplot object showing low-dimensional embedding of subjects.

Examples

```
## Not run:  
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)  
plot_cluster_embedding(sim, method = "umap")  
plot_cluster_embedding(sim, method = "pca")  
  
## End(Not run)
```

`plot_cluster_synergy_loadings_mfa`*Plot Cluster-Specific Synergy Coefficients (Factor Loadings) for MFA*

Description

This function takes a fitted MFA model (e.g., from `mfa_em_fit()`) and produces a faceted heatmap or bar plot of each cluster's factor loadings.

Usage

```
plot_cluster_synergy_loadings_mfa(  
  mfa_fit,  
  cluster_ids = NULL,  
  plot_type = c("heatmap", "bar"),  
  muscle_names = NULL  
)
```

Arguments

<code>mfa_fit</code>	A fitted MFA model object containing Lambda, z, etc.
<code>cluster_ids</code>	A numeric vector of cluster IDs to plot. Defaults to all.
<code>plot_type</code>	One of "heatmap" or "bar". Specifies how to visualize.
<code>muscle_names</code>	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object (which you can print or further modify).

Examples

```
## Not run:  
# Suppose we have a best MFA model in 'best_mfa_model'  
p <- plot_cluster_synergy_loadings_mfa(best_mfa_model, cluster_ids = 1:2)  
print(p)  
  
## End(Not run)
```

plot_cluster_synergy_loadings_mPCA

Plot Cluster-Specific Synergy Coefficients (Factor Loadings) for MPCA

Description

This function takes a fitted MPCA model (e.g. from `mixture_pca_em_fit()`) and produces a faceted heatmap or bar plot of each cluster's factor loadings $W[[k]]$ (an $M \times r$ matrix).

Usage

```
plot_cluster_synergy_loadings_mPCA(
  mpca_fit,
  cluster_ids = NULL,
  plot_type = c("heatmap", "bar"),
  muscle_names = NULL
)
```

Arguments

<code>mpca_fit</code>	A fitted MPCA model object containing W , z , etc.
<code>cluster_ids</code>	A numeric vector of cluster IDs to plot. Defaults to all.
<code>plot_type</code>	One of "heatmap" or "bar". Specifies how to visualize.
<code>muscle_names</code>	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object (which you can print or further modify).

Examples

```
## Not run:
# Suppose we have a best MPCA model in 'best_mpca_model'
# that has best_mpca_model$W[[k]] for each cluster
p <- plot_cluster_synergy_loadings_mPCA(best_mpca_model, cluster_ids = 1:2)
print(p)

## End(Not run)
```

plot_confusion_matrix *Plot Confusion Matrix for Cluster Assignments*

Description

Creates a confusion matrix heatmap comparing true cluster assignments with estimated cluster assignments. This is the standard way to visualize clustering accuracy and identify systematic misclassification patterns.

Usage

```
plot_confusion_matrix(z_true, z_est, normalize = "true", theme_base_size = 14)
```

Arguments

z_true	Vector of true cluster assignments (length N, values 1 to K).
z_est	Vector of estimated cluster assignments (length N, values 1 to K).
normalize	Character string specifying normalization: "none" (raw counts), "true" (normalize by true cluster size), or "pred" (normalize by predicted cluster size). Default is "true".
theme_base_size	Base font size for the plot theme. Default is 14.

Details

The confusion matrix shows the correspondence between true and estimated cluster assignments. Rows represent true clusters, columns represent estimated clusters. Cell values indicate the number (or proportion) of subjects with that true-estimated cluster pair.

Perfect clustering produces a diagonal matrix (all subjects correctly assigned). Off-diagonal entries indicate misclassifications. This visualization complements the ARI metric by showing:

- Which clusters are confused with each other
- Whether misclassifications are systematic or random
- Cluster size imbalances
- Label permutation issues (if estimated clusters are numbered differently)

Normalization options:

- "none": Raw counts (useful for seeing absolute numbers)
- "true": Divide by true cluster size (shows recall/sensitivity)
- "pred": Divide by predicted cluster size (shows precision)

Value

A ggplot object showing the confusion matrix as a heatmap.

Examples

```
## Not run:
# Simulate data
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)

# Fit Two-step baseline
fit <- fit_two_step_baseline(
  sim$list_of_data, r=3, K=3, method="PCA",
  z_true=sim$z_true, align_basis=TRUE
)

# Plot confusion matrix
plot_confusion_matrix(sim$z_true, fit$z_est)

# Show raw counts instead of proportions
plot_confusion_matrix(sim$z_true, fit$z_est, normalize="none")

# Normalize by predicted cluster size
plot_confusion_matrix(sim$z_true, fit$z_est, normalize="pred")

## End(Not run)
```

plot_fig1_ARI

*Plot ARI vs. Separation/Noise (Figure 1)***Description**

Creates a faceted plot showing ARI (Adjusted Rand Index) values for MFA and MPCA across different separation and noise conditions.

Usage

```
plot_fig1_ARI(df)
```

Arguments

df A data.frame with columns K_true, r_true, N, M, sep, noise, ARI_MFA, ARI_PCA.

Value

A ggplot object.

Examples

```
## Not run:
df <- read_all_results("results")
p <- plot_fig1_ARI(df)
print(p)
ggsave("figure1_ARI.pdf", p, width = 8, height = 6)

## End(Not run)
```

plot_fig1_baseline_comparison

Plot Baseline Comparison (ARI and VAF)

Description

Creates a bar plot comparing ARI and VAF values across different methods, including Two-step baselines. This visualization highlights the performance differences between single-model approaches, two-step baselines, and mixture models.

Usage

```
plot_fig1_baseline_comparison(df, metric = c("ARI", "VAF"))
```

Arguments

df	A data.frame with columns Method, ARI, VAF. Typically the output from simulate_and_compare_5me
metric	Character string specifying which metric to plot: "ARI" or "VAF". Defaults to "ARI".

Details

The plot uses color coding to distinguish between method types:

- Single methods (SingleFA, SinglePCA) - shown in one color group
- Two-step methods (TwoStep_FA, TwoStep_PCA) - shown in another color group
- Mixture methods (MixtureFA, MixturePCA) - shown in a third color group

This visualization is designed to demonstrate the superiority of mixture-based approaches over simpler baseline methods.

Value

A ggplot object showing a bar plot of the specified metric by method.

Examples

```
## Not run:
df_comp <- simulate_and_compare_5methods(
  N = 100, K_true = 3, r_true = 4, M = 6, T_each = 100
)
p_ari <- plot_fig1_baseline_comparison(df_comp, metric = "ARI")
p_vaf <- plot_fig1_baseline_comparison(df_comp, metric = "VAF")
print(p_ari)
ggsave("baseline_comparison_ARI.pdf", p_ari, width = 8, height = 6)

## End(Not run)
```

plot_fig2_BIC_heatmap *Plot BIC Heatmap for a Subset (Figure 2)*

Description

Creates a heatmap of BIC values for a specific subset of conditions.

Usage

```
plot_fig2_BIC_heatmap(df)
```

Arguments

df A data.frame with columns including N, M, sep, noise, K_true, r_true, BIC_MFA.

Value

A ggplot object.

Examples

```
## Not run:
df <- read_all_results("results")
p <- plot_fig2_BIC_heatmap(df)
print(p)
ggsave("figure2_BIC.pdf", p, width = 6, height = 5)

## End(Not run)
```

`plot_fig3_init_boxplot`*Plot Boxplot of ARI for Multiple Initial Seeds (Figure 3)*

Description

Creates a boxplot showing the distribution of ARI values across multiple random initializations.

Usage

```
plot_fig3_init_boxplot(df)
```

Arguments

`df` A `data.frame` with columns `ARI_MFA`, `ARI_PCA`.

Value

A `ggplot` object.

Examples

```
## Not run:
df <- read_all_results("results")
p <- plot_fig3_init_boxplot(df)
print(p)
ggsave("figure3_boxplot.pdf", p, width = 6, height = 4)

## End(Not run)
```

`plot_fig4to6_reconstruction_vaf`*Plot Reconstruction Error and VAF (Figures 4-6)*

Description

Creates visualizations of reconstruction error (SSE) and Variance Accounted For (VAF) for different methods and conditions.

Usage

```
plot_fig4to6_reconstruction_vaf(df)
```

Arguments

`df` A `data.frame` with columns including VAF and SSE metrics.

Value

A ggplot object.

Examples

```
## Not run:
df <- read_all_results("results")
p <- plot_fig4to6_reconstruction_vaf(df)
print(p)
ggsave("figure4to6_VAF.pdf", p, width = 10, height = 8)

## End(Not run)
```

plot_metric_vs_axis *Plot Metric vs Axis for Parameter Sweep*

Description

Creates a line plot showing how a performance metric varies with a parameter axis (e.g., cluster separation). Useful for exploring parameter sensitivity.

Usage

```
plot_metric_vs_axis(
  aggregated_df,
  metric = "ARI",
  axis = "cluster_sep_spatial",
  facet_by = "model_type",
  color_by = "Method",
  show_errorBars = TRUE
)
```

Arguments

aggregated_df	Data frame with aggregated results from run_param_sweep(). Must contain columns: Method, model_type, the specified metric columns (metric_mean, metric_sd), and the specified axis column.
metric	Character. Metric to plot. One of "ARI", "VAF", "BIC", "SSE". Default is "ARI".
axis	Character. Parameter axis to plot on x-axis. One of "cluster_sep_spatial", "cluster_sep_temporal", "cluster_sep_stability", "N", "K", "r". Default is "cluster_sep_spatial".
facet_by	Character. Variable to facet by. One of "model_type" or "Method". Default is "model_type".
color_by	Character. Variable to color by. One of "Method" or "model_type". Default is "Method".
show_errorBars	Logical. Whether to show error bars (mean +/- sd). Default is TRUE.

Value

A ggplot2 object

Examples

```
## Not run:
library(synergyMixR)
results <- run_param_sweep(test = TRUE, cores = 2)

# Plot ARI vs spatial separation
p1 <- plot_metric_vs_axis(
  results$aggregated_results,
  metric = "ARI",
  axis = "cluster_sep_spatial"
)
print(p1)

# Plot VAF vs temporal separation
p2 <- plot_metric_vs_axis(
  results$aggregated_results,
  metric = "VAF",
  axis = "cluster_sep_temporal"
)
print(p2)

## End(Not run)
```

plot_neural_command *Plot Neural Command Trajectories*

Description

Visualizes time-varying neural command trajectories estimated from synergy activation patterns. This function creates a low-dimensional representation of the temporal evolution of neural control signals, revealing how motor commands change over time within and across clusters.

Usage

```
plot_neural_command(
  sim,
  method = "pca",
  color_by = "cluster",
  smooth = TRUE,
  theme_base_size = 14
)
```

Arguments

sim	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>C_list</code> and <code>z_true</code> components.
method	Character string specifying the embedding method: "pca" or "umap". Default is "pca".
color_by	Character string specifying how to color points: "cluster" (by cluster assignment) or "time" (by temporal progression). Default is "cluster".
smooth	Logical; if TRUE, connect time points with smooth paths using <code>geom_path</code> with <code>alpha gradient</code> . Default is TRUE.
theme_base_size	Base font size for the plot theme. Default is 14.

Value

A ggplot object showing neural command trajectories in 2D space.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_neural_command(sim, method = "pca", color_by = "cluster")
plot_neural_command(sim, method = "umap", color_by = "time", smooth = TRUE)

## End(Not run)
```

plot_noise_profiles *Plot Noise Profiles*

Description

Visualizes the distribution of noise parameters across clusters, showing how stability-adaptability varies between clusters.

Usage

```
plot_noise_profiles(sim, theme_base_size = 14)
```

Arguments

sim	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>stability_params</code> component.
theme_base_size	Base font size for the plot theme. Default is 14.

Value

A ggplot object showing noise parameter distributions per cluster.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_noise_profiles(sim)

## End(Not run)
```

plot_reconstructed_emg

Plot Reconstructed EMG Signals

Description

Visualizes EMG time-series per cluster and subject. When `cluster_id` is `NULL`, shows all clusters. For each cluster, individual subject EMG traces are shown as thin semi-transparent lines, with the cluster-mean waveform overlaid as a bold black line.

Usage

```
plot_reconstructed_emg(
  sim,
  cluster_id = NULL,
  muscles = NULL,
  mean_ci = FALSE,
  theme_base_size = 13
)
```

Arguments

<code>sim</code>	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>X_list</code> and <code>z_true</code> components.
<code>cluster_id</code>	Integer specifying which cluster to plot. If <code>NULL</code> , plots all clusters. Default is <code>NULL</code> .
<code>muscles</code>	Integer vector specifying which muscles to plot. If <code>NULL</code> , plots all muscles. Default is <code>NULL</code> .
<code>mean_ci</code>	Logical; if <code>TRUE</code> , display ± 1 SD ribbons (not yet implemented). Default is <code>FALSE</code> .
<code>theme_base_size</code>	Base font size for the plot theme. Default is 13.

Value

A ggplot object showing EMG time-series with individual subjects and cluster means.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_reconstructed_emg(sim)
plot_reconstructed_emg(sim, cluster_id = 1)
plot_reconstructed_emg(sim, muscles = c(1, 2, 3))

## End(Not run)
```

plot_sse_heatmap	<i>Plot SSE Heatmap Across Methods and Subjects</i>
------------------	---

Description

Creates a heatmap showing Sum of Squared Errors (SSE) for each subject across different methods. This helps identify which methods perform poorly for specific subjects and reveals the instability of the Two-step approach.

Usage

```
plot_sse_heatmap(
  results_list,
  list_of_data,
  log_scale = TRUE,
  theme_base_size = 12
)
```

Arguments

results_list	A named list of method results. Each element should be a fitted model object containing SSE information. Names should be method names (e.g., "TwoStep_PCA", "MixtureFA", "SinglePCA").
list_of_data	List of data matrices, one per subject ($T_i \times M$). Required to compute per-subject SSE if not already in results.
log_scale	Logical. If TRUE, displays $\log_{10}(\text{SSE})$ instead of raw SSE. Recommended for better visualization when SSE values span multiple orders of magnitude. Default is TRUE.
theme_base_size	Base font size for the plot theme. Default is 12.

Details

This function computes per-subject reconstruction errors for multiple methods and displays them as a heatmap. Darker colors indicate higher errors (worse reconstruction). This visualization is useful for:

- Comparing reconstruction quality across methods
- Identifying subjects that are poorly reconstructed by Two-step methods
- Revealing instability in Two-step assignments (high SSE for misclassified subjects)
- Showing that mixture models provide more consistent reconstruction

The function expects results from `fit_all_methods_on_data()` or similar comparison functions. It computes per-subject SSE by reconstructing each subject's data using the assigned cluster model.

Value

A ggplot object showing a heatmap of SSE values (subjects x methods).

Examples

```
## Not run:
# Simulate data
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)

# Fit all methods
results <- fit_all_methods_on_data(
  sim$list_of_data, z_true=sim$z_true,
  N=30, K=3, r=3, M=8, seed=123
)

# Create SSE heatmap
plot_sse_heatmap(results, sim$list_of_data)

# Use linear scale instead of log scale
plot_sse_heatmap(results, sim$list_of_data, log_scale=FALSE)

## End(Not run)
```

plot_sweep_summary

Plot Summary Comparison for Parameter Sweep

Description

Creates a bar plot comparing method performance across all parameter combinations, showing mean performance with error bars.

Usage

```
plot_sweep_summary(
  aggregated_df,
  metric = "ARI",
  facet_by = "model_type",
  summary = "mean_sd"
)
```

Arguments

aggregated_df	Data frame with aggregated results from run_param_sweep(). Must contain columns: Method, model_type, and the specified metric columns.
metric	Character. Metric to plot. One of "ARI", "VAF", "BIC", "SSE". Default is "ARI".
facet_by	Character. Variable to facet by. One of "model_type" or "Method". Default is "model_type".
summary	Character. Type of summary to show. One of "mean_sd" (mean +/- sd) or "min_max" (min to max range). Default is "mean_sd".

Value

A ggplot2 object

Examples

```
## Not run:
library(synergyMixR)
results <- run_param_sweep(test = TRUE, cores = 2)

# Plot overall ARI comparison
p <- plot_sweep_summary(results$aggregated_results, metric = "ARI")
print(p)

## End(Not run)
```

plot_synergy_bases	<i>Plot Synergy Bases (Lambda Matrices)</i>
--------------------	---

Description

Creates heatmaps of synergy matrices (Lambda_k) for each cluster. Each heatmap shows the muscle-synergy structure with muscles on the y-axis and synergies on the x-axis.

Usage

```
plot_synergy_bases(sim, theme_base_size = 14)
```

Arguments

sim	A list returned by <code>simulate_dynamic_synergy_data()</code> . Must contain <code>Lambda_list</code> component.
theme_base_size	Base font size for the plot theme. Default is 14.

Value

A ggplot object showing heatmaps of synergy matrices per cluster.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_synergy_bases(sim)

## End(Not run)
```

plot_synergy_correlation_comparison

Plot Synergy Correlation Comparison

Description

Compares true synergies with estimated synergies by computing and visualizing the correlation matrix between them. This extends the existing `plot_synergy_correlation()` function to compare true vs estimated synergies rather than comparing between clusters.

Usage

```
plot_synergy_correlation_comparison(
  W_true,
  W_est,
  z_true,
  z_est,
  cluster_id = NULL,
  method = "pearson",
  theme_base_size = 12
)
```

Arguments

W_true	List of true basis matrices (e.g., <code>sim\$Lambda_list</code>). Each matrix should be $M \times r$.
W_est	List of estimated basis matrices (e.g., from fitted model). Each matrix should be $M \times r$.

<code>z_true</code>	Vector of true cluster assignments (length N).
<code>z_est</code>	Vector of estimated cluster assignments (length N).
<code>cluster_id</code>	Integer specifying which cluster to analyze. If NULL, analyzes all clusters separately. Default is NULL.
<code>method</code>	Correlation method: "pearson", "kendall", or "spearman". Default is "pearson".
<code>theme_base_size</code>	Base font size for the plot theme. Default is 12.

Details

This function computes the correlation between true synergy patterns (from simulation data) and estimated synergy patterns (from fitted models). For each cluster, it creates a correlation matrix where:

- Rows represent true synergies (components)
- Columns represent estimated synergies (components)
- Cell values show correlation between true component *i* and estimated component *j*

Perfect recovery produces a diagonal matrix with high correlations (close to 1) on the diagonal and low correlations off-diagonal. This indicates that each estimated component correctly matches its corresponding true component.

Common failure patterns:

- Permuted components: High correlations off-diagonal (alignment failure)
- Low correlations everywhere: Poor decomposition quality
- Mixed patterns: Some components recovered, others not

Value

A ggplot object showing correlation heatmap(s) between true and estimated synergies.

Examples

```
## Not run:
# Simulate data
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)

# Fit Two-step baseline
fit <- fit_two_step_baseline(
  sim$list_of_data, r=3, K=3, method="PCA",
  z_true=sim$z_true, align_basis=TRUE
)

# Compare true and estimated synergies
plot_synergy_correlation_comparison(
  sim$lambda_list, fit$W_list,
  sim$z_true, fit$z_est
)
```

```
# Analyze specific cluster
plot_synergy_correlation_comparison(
  sim$Lambda_list, fit$W_list,
  sim$z_true, fit$z_est,
  cluster_id = 1
)

## End(Not run)
```

plot_synergy_summary *Plot Synergy Summary*

Description

Creates a comprehensive summary plot combining multiple visualization panels using patchwork. This provides an overview of spatial synergy structures, temporal activation dynamics, and cluster separability.

Usage

```
plot_synergy_summary(sim, fit = NULL, theme_base_size = 12)
```

Arguments

sim	A list returned by <code>simulate_dynamic_synergy_data()</code> .
fit	Optional fitted model object (e.g., from MFA/MPCA). Default is NULL.
theme_base_size	Base font size for the plot theme. Default is 12.

Value

A patchwork object combining multiple ggplot panels.

Examples

```
## Not run:
sim <- simulate_dynamic_synergy_data(N=30, K=3, r=3, M=8, T_each=200)
plot_synergy_summary(sim)

## End(Not run)
```

plot_waveform_comparison_mfa

Plot Original vs. Reconstructed Waveform for an MFA Model

Description

Plot Original vs. Reconstructed Waveform for an MFA Model

Usage

```
plot_waveform_comparison_mfa(
  mfa_fit,
  list_of_data,
  subject_id,
  channel = 1,
  muscle_names = NULL
)
```

Arguments

mfa_fit	A fitted MFA model with Lambda, Psi, mu, z.
list_of_data	The original data list used to fit the model.
subject_id	Index of the subject to plot.
channel	Which channel (muscle) to plot. Defaults to 1.
muscle_names	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object showing original and reconstructed waveforms.

plot_waveform_comparison_mPCA

Plot Original vs. Reconstructed Waveform for an MPCA Model

Description

Plot Original vs. Reconstructed Waveform for an MPCA Model

Usage

```
plot_waveform_comparison_mpca(
  mpca_fit,
  list_of_data,
  subject_id,
  channel = 1,
  muscle_names = NULL
)
```

Arguments

mpca_fit	A fitted MPCA model with W, mu, z, plus factor_scores.
list_of_data	The original data list used to fit the model.
subject_id	Index of the subject to plot (1-based).
channel	Which channel (1-based) to plot. Defaults to 1.
muscle_names	Optional character vector of muscle names for y-axis labels. If NULL, default channel names will be used.

Value

A ggplot object showing original and reconstructed waveforms.

plot_winrate_heatmap *Plot Win Rate Heatmap for Parameter Sweep*

Description

Creates a heatmap showing the percentage of parameter combinations where each method performed best for each metric.

Usage

```
plot_winrate_heatmap(win_rates_df, facet_by = "metric")
```

Arguments

win_rates_df	Data frame with win rates from run_param_sweep(). Must contain columns: Method, model_type, metric, win_rate. This is typically the data from the win_rates.csv file.
facet_by	Character. Variable to facet by. One of "model_type" or "metric". Default is "metric".

Value

A ggplot2 object

Examples

```
## Not run:
library(synergyMixR)
results <- run_param_sweep(test = TRUE, cores = 2)

# Read win rates
win_rates <- read.csv(results$win_rates_file)

# Plot win rate heatmap
p <- plot_winrate_heatmap(win_rates)
print(p)

## End(Not run)
```

posthoc_mfa_evaluation

Post-hoc Evaluation of MFA Grid Search

Description

Given the output of `select_optimal_K_r_mfa`, this function computes additional metrics for each fitted model in `all_models`, such as global VAF (`compute_global_vaf_mfa`) and cluster sizes (`compute_cluster_sizes`).

Usage

```
posthoc_mfa_evaluation(list_of_data, selection_obj)
```

Arguments

`list_of_data` The same list of data used to fit the models.
`selection_obj` The object returned by `select_optimal_K_r_mfa`.

Details

This function loops over each entry in `selection_obj$all_models`, extracts the model, calls `compute_global_vaf_mfa` and `compute_cluster_sizes`, and records the results in a combined table.

Value

A data frame with columns:

K	Number of clusters.
r	Factor dimension.
logLik	Log-likelihood from the fit.

BIC	BIC from the fit.
GlobalVAF	Global variance accounted for, $1 - \text{SSE}/\text{SST}$.
minClusterSize	Minimum cluster size.
maxClusterSize	Maximum cluster size.

See Also

[select_optimal_K_r_mfa](#), [compute_factor_scores](#)

Examples

```
## Not run:
sel_obj <- select_optimal_K_r_mfa(my_data, Kvec=2:4, rvec=1:3)
df_posthoc <- posthoc_mfa_evaluation(my_data, sel_obj)
head(df_posthoc)

## End(Not run)
```

posthoc_mPCA_evaluation

Post-hoc Evaluation of Mixture PCA Grid Search

Description

Takes the output of [select_optimal_K_r_mPCA](#) and computes additional metrics (global VAF, cluster sizes, etc.) for each entry in `all_models`.

Usage

```
posthoc_mPCA_evaluation(list_of_data, selection_obj)
```

Arguments

<code>list_of_data</code>	The same list of data used for the grid search.
<code>selection_obj</code>	The object returned by select_optimal_K_r_mPCA .

Value

A data frame with columns `K`, `r`, `logLik`, `BIC`, `GlobalVAF`, `minClusterSize`, `maxClusterSize`.

preprocess_emg	<i>EMG Preprocessing Pipeline</i>
----------------	-----------------------------------

Description

Applies a complete preprocessing pipeline to raw EMG data, transforming it into the format required for MFA/MPCA analysis.

Usage

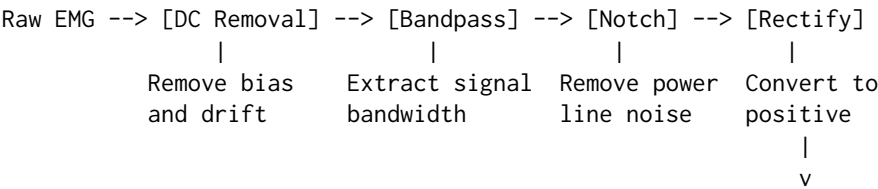
```
preprocess_emg(  
  raw_data,  
  sampling_rate,  
  pipeline_config = "default",  
  mvc_data = NULL,  
  channel_names = NULL,  
  verbose = TRUE  
)
```

Arguments

raw_data	Raw EMG data. Can be: • A matrix (T x M) for a single trial • A list of matrices for multiple trials/subjects
sampling_rate	Sampling frequency in Hz.
pipeline_config	Preprocessing configuration. Can be: • A character string: preset name ("default", "gait", "isometric", "fine_motor") • A list: custom configuration (see Details)
mvc_data	Optional MVC reference data for MVC normalization. Can be a matrix (T x M) or a list of matrices.
channel_names	Optional character vector of channel/muscle names.
verbose	Logical. If TRUE (default), prints progress information.

Details

Processing Flow: The preprocessing pipeline transforms raw EMG signals into activation patterns suitable for muscle synergy analysis:



```

Preprocessed <-- [Normalize] <-- [Envelope] <-----+
(list_of_data)      |              |
                   Scale for      Extract amplitude
                   comparison      modulation

```

Processing Steps:

- 1. DC Offset Removal** Removes electrode DC bias and signal drift. Methods: "mean" (subtract mean), "linear" (detrend), "polynomial". Typical use: Always recommended as first step.
- 2. Bandpass Filter** Extracts the EMG frequency band (typically 20-450 Hz). Removes low-frequency motion artifacts (<20 Hz) and high-frequency noise (>450 Hz). Parameters: low_cutoff=20, high_cutoff=450, order=4 (Butterworth).
- 3. Notch Filter** Removes power line interference (50 Hz in Japan/Europe, 60 Hz in Americas). Can also remove harmonics (100, 150, 200 Hz, etc.). Parameters: freq=50 or 60, Q=30, harmonics=TRUE.
- 4. Rectification** Converts bipolar EMG to unipolar by taking absolute value (full-wave) or setting negatives to zero (half-wave). Full-wave rectification is standard for envelope extraction.
- 5. Envelope Extraction** Extracts the amplitude envelope representing muscle activation pattern. Methods: "lowpass" (4-10 Hz cutoff), "rms" (root mean square), "moving_average". Low-pass with 6 Hz cutoff is most common.
- 6. Normalization** Scales signals for comparison across muscles/subjects/sessions. Methods: "mvc" (percentage of max voluntary contraction - gold standard), "max" (peak value), "zscore" (standardization), "range" (0-1 scaling).

Preset Configurations:

- default** General-purpose settings. Bandpass 20-450 Hz, 50 Hz notch, lowpass envelope (6 Hz), max normalization.
- gait** Optimized for walking/gait analysis. Linear detrend, slower envelope (4 Hz cutoff), MVC normalization.
- isometric** For sustained contractions. RMS envelope (100 ms window), MVC normalization. No time normalization needed.
- fine_motor** For precise movements. Wider bandwidth (10-500 Hz), faster envelope (10 Hz cutoff).

Custom Configuration:

```

config <- list(
  dc_removal = list(method = "mean"),
  bandpass = list(low = 20, high = 450, order = 4),
  notch = list(freq = 50, harmonics = TRUE),
  rectify = list(method = "full_wave"),
  envelope = list(method = "lowpass", cutoff = 6),
  normalize = list(method = "max")
)

```

Set any step to NULL or FALSE to skip it.

Value

An object of class "preprocessed_emg" containing:

- data: Preprocessed data in list_of_data format (ready for MFA/MPCA)
- processing_info: Record of all preprocessing steps and parameters
- channel_names: Channel names
- sampling_rate: Original sampling rate

See Also

[emg_preset_config](#) for preset configurations, [bandpass_filter](#), [notch_filter](#), [rectify](#), [extract_envelope](#), [normalize_emg](#) for individual processing functions.

Examples

```
## Not run:
# Basic usage with default settings
raw_emg <- matrix(rnorm(1000 * 8), ncol = 8)
processed <- preprocess_emg(raw_emg, sampling_rate = 1000)

# Use with MFA analysis
fit <- mfa_em_fit(processed$data, K = 2, r = 3)

# Custom configuration
config <- list(
  bandpass = list(low = 30, high = 400),
  notch = list(freq = 60), # 60 Hz for Americas
  envelope = list(method = "rms", window_ms = 50),
  normalize = list(method = "mvc")
)
processed <- preprocess_emg(raw_emg, sampling_rate = 1000,
  pipeline_config = config,
  mvc_data = mvc_trials)

## End(Not run)
```

```
print.bootstrap_ci      Print Summary of Bootstrap CI Results
```

Description

Print Summary of Bootstrap CI Results

Usage

```
## S3 method for class 'bootstrap_ci'
print(x, cluster = 1, ...)
```

Arguments

x	Result from bootstrap_loading_ci
cluster	Which cluster to summarize (default: 1)
...	Additional arguments (ignored)

```
print.calibration_result
```

Print Summary of Calibration Results

Description

Print Summary of Calibration Results

Usage

```
## S3 method for class 'calibration_result'  
print(x, ...)
```

Arguments

x	A calibration_result object
...	Additional arguments (ignored)

```
print.mixture_r_selection
```

Print Method for mixture_r_selection Objects

Description

Print Method for mixture_r_selection Objects

Usage

```
## S3 method for class 'mixture_r_selection'  
print(x, ...)
```

Arguments

x	A mixture_r_selection object.
...	Additional arguments (ignored).

```
print.preprocessed_emg
```

Print Method for preprocessed_emg Objects

Description

Print Method for preprocessed_emg Objects

Usage

```
## S3 method for class 'preprocessed_emg'
print(x, ...)
```

Arguments

x	A preprocessed_emg object
...	Additional arguments (ignored)

```
read_all_results
```

Read All Results from .rds Files

Description

This function searches a directory for .rds files, each presumably containing one or more rows of simulation results, then merges them into a single data frame. Each row includes a file column indicating the source file path.

Usage

```
read_all_results(result_dir = "results")
```

Arguments

result_dir	A character string specifying the directory to look for .rds files. Defaults to "results".
------------	--

Details

If no files are found, the function raises an error. Otherwise, each file is read with [readRDS](#) and merged using `dplyr::bind_rows`, which safely handles heterogeneous column structures by filling missing columns with NA.

Value

A combined data.frame with rows from all loaded files, including a file column.

Examples

```
## Not run:  
df <- read_all_results("results")  
head(df)  
  
## End(Not run)
```

`read_and_visualize_all`*Read All Results and Produce Tables/Figures*

Description

This function reads all .rds result files in a specified directory, merges them into a single data frame, prints summary tables, and generates several figures.

Usage

```
read_and_visualize_all(result_dir = "results")
```

Arguments

<code>result_dir</code>	A character string specifying the directory containing .rds result files. Defaults to "results".
-------------------------	--

Value

Invisibly returns the merged data frame of all results.

Examples

```
## Not run:  
df_all <- read_and_visualize_all("results")  
head(df_all)  
  
## End(Not run)
```

reconstruct_singleMPCA	<i>Reconstruct a single subject's data from a Mixture PCA model</i>
------------------------	---

Description

Reconstruct a single subject's data from a Mixture PCA model

Usage

reconstruct_singleMPCA(X, mu, W, Z)

Arguments

X	Original (T x M) data
mu	An (M) mean vector
W	An (M x r) loading matrix
Z	A (T x r) factor score matrix

Value

A (T x M) reconstructed matrix Xhat

rectify	<i>Rectify EMG Signal</i>
---------	---------------------------

Description

Applies rectification to EMG signals, converting all values to positive.

Usage

rectify(data, method = "full_wave")

Arguments

data	EMG signal. Can be a numeric vector (single channel), a matrix (T x M, time x channels), or a list of matrices.
method	Rectification method: • "full_wave": Take absolute value (default) • "half_wave": Set negative values to zero

Details

Rectification is a standard step in EMG processing that prepares the signal for envelope extraction. Full-wave rectification (absolute value) is most commonly used.

Value

Rectified EMG signal.

Examples

```
signal <- rnorm(1000)
rectified <- rectify(signal) # full-wave rectification
half_rect <- rectify(signal, method = "half_wave")
```

remove_dc_offset	<i>Remove DC Offset from EMG Signal</i>
------------------	---

Description

Removes the DC component (baseline offset) from EMG signals using various methods.

Usage

```
remove_dc_offset(data, method = "mean", poly_order = 1)
```

Arguments

data	EMG signal. Can be a numeric vector (single channel), a matrix (T x M, time x channels), or a list of matrices.
method	Method for DC removal: • "mean": Subtract the mean (default) • "median": Subtract the median (robust to outliers) • "linear": Remove linear trend (detrend) • "polynomial": Remove polynomial trend
poly_order	Polynomial order when method = "polynomial". Default is 1 (equivalent to linear detrending).

Value

Data with DC offset removed, same structure as input.

Examples

```
# Single channel
signal <- rnorm(1000) + 0.5
clean <- remove_dc_offset(signal)

# Multi-channel matrix
emg <- matrix(rnorm(1000 * 8), nrow = 1000, ncol = 8)
emg_clean <- remove_dc_offset(emg, method = "linear")
```

rotate_mfa_model

Post-hoc Rotation for an MFA Model

Description

Applies a chosen rotation (e.g., "varimax") to each cluster's loading matrix (`Lambda[[k]]`). Optionally, also rotates the factor scores in `factor_scores[[i]]` if they exist and `rotate_scores=TRUE`.

Usage

```
rotate_mfa_model(
  mfa_fit,
  rotation = c("varimax"),
  rotate_scores = FALSE,
  cluster_ids = NULL
)
```

Arguments

<code>mfa_fit</code>	A fitted MFA model, typically returned by <code>mfa_em_fit()</code> or <code>select_optimal_K_r_mfa()</code> \$best_model, containing: • <code>Lambda[[k]]</code>: (M x r) factor loadings for each cluster. • <code>factor_scores[[i]]</code>: optional (T_i x r) factor scores for subject i. • <code>z[i]</code>: cluster assignments.
<code>rotation</code>	A string specifying the rotation method. Currently supports "varimax".
<code>rotate_scores</code>	Logical; if TRUE and factor_scores exist, rotate them as well. If factor_scores do not exist and rotate_scores=TRUE, an error or warning is produced.
<code>cluster_ids</code>	Vector of cluster indices to rotate. Defaults to all.

Details

- Rotation does not change log-likelihood or BIC, just reorients factors for interpretability. - If factor_scores is missing and rotate_scores=TRUE, you can either skip or error.

Value

A new MFA model object, with rotated Lambda (and factor_scores if requested). Other fields are unchanged.

rotate_mpca_model	<i>Post-hoc Rotation for a Mixture PCA Model</i>
-------------------	--

Description

Applies a chosen rotation (e.g. varimax) to each cluster's loading matrix $W[[k]]$ in a fitted MPCA model, optionally also rotating the subject factor scores in $\text{factor_scores}[[i]]$.

Usage

```
rotate_mpca_model(
  mpca_fit,
  rotation = c("varimax"),
  rotate_scores = FALSE,
  cluster_ids = NULL
)
```

Arguments

mpca_fit	A fitted MPCA model, e.g. returned by <code>mixture_pca_em_fit()</code> or <code>select_optimal_K_r_mpca()</code> \$best, typically containing: • $W[[k]]$: ($M \times r$) loadings for cluster k • $\text{factor_scores}[[i]]$: optional ($T_i \times r$) PC scores for subject i • $z[i]$: cluster assignments
rotation	Rotation method, e.g. "varimax".
rotate_scores	Logical; if TRUE, also rotate factor_scores if present.
cluster_ids	Vector of cluster indices to rotate. Defaults to all.

Details

In R's `stats::varimax()`, the matrix should have rows = variables, cols = factors. If W_k is already ($M \times r$), then passing it directly to `varimax` means we have M variables, r factors. The returned rotation matrix is ($r \times r$), and scores_i ($T_i \times r$) can be multiplied by that.

Value

A new MPCA model object with $W[[k]]$ rotated, and factor_scores also rotated if requested and available.

run_k_selection_comparison

Run K Selection Method Comparison (v3)

Description

Compares multiple K selection methods (BIC, ICL with various temperatures, Ensemble, Adaptive, Silhouette) across a parameter grid.

Usage

```
run_k_selection_comparison(
  test = FALSE,
  cores = 4,
  output_dir = NULL,
  parallel_strategy = c("local", "pbs", "future"),
  pbs_config = list(),
  pbs_submit = FALSE,
  pbs_wait = FALSE,
  Kvec = 2:6,
  r_fixed = 2,
  max_iter = 50,
  verbose = TRUE
)
```

Arguments

test	Logical. If TRUE, use smaller test grid. Default FALSE.
cores	Integer. Number of CPU cores. Default 4.
output_dir	Character. Output directory. Default auto-detected.
parallel_strategy	Character. "local", "pbs", or "future".
pbs_config	List. PBS configuration (queue, walltime, mem, r_module).
pbs_submit	Logical. If TRUE, submit PBS job. Default FALSE (dry run).
pbs_wait	Logical. If TRUE, wait for PBS completion. Default FALSE.
Kvec	Integer vector. Candidate K values. Default 2:6.
r_fixed	Integer. Fixed r value for two-stage. Default 2.
max_iter	Integer. Maximum EM iterations. Default 50.
verbose	Logical. Print progress. Default TRUE.

Details

This function compares the following K selection methods:

- BIC: Bayesian Information Criterion
- ICL_T1: ICL with temperature = 1.0
- ICL_T2: ICL with temperature = 2.0 (recommended)
- ICL_T3: ICL with temperature = 3.0
- Ensemble: Majority vote of BIC, ICL_T1, ICL_T2
- Adaptive: Adaptive temperature selection
- Silhouette: Silhouette-based selection

Value

Data frame with results, or PBS job info if using PBS.

Examples

```
## Not run:
# Local test run
results <- run_k_selection_comparison(test = TRUE, cores = 2)

# Full simulation
results <- run_k_selection_comparison(test = FALSE, cores = 8)

# PBS submission
run_k_selection_comparison(
  test = FALSE,
  parallel_strategy = "pbs",
  pbs_config = list(queue = "batch"),
  pbs_submit = TRUE
)

## End(Not run)
```

run_param_sweep

Run Parameter Sweep Experiment

Description

Orchestrates a comprehensive parameter sweep experiment to evaluate the performance of MFA, MPCA, and baseline methods using the v2 dynamic synergy simulator with three orthogonal control axes.

Usage

```
run_param_sweep(
  test = FALSE,
  cores = 4,
  output_dir = NULL,
  param_grid = NULL,
  verbose = TRUE
)
```

Arguments

test	Logical. If TRUE, uses a smaller test grid for quick validation. Default is FALSE (full parameter grid).
cores	Integer. Number of CPU cores to use for parallel processing. Default is 4. Set to 1 for sequential processing.
output_dir	Character. Directory to save results. Default is "inst/paper/results_sweep" if it exists (repo use), otherwise "results_sweep" in the current working directory.
param_grid	Optional. Custom parameter grid data frame. If NULL, uses the built-in full or test grid based on the test parameter.
verbose	Logical. If TRUE, prints progress information. Default is TRUE.

Details

The parameter sweep evaluates:

- 6 methods: SingleFA, SinglePCA, TwoStep_FA, TwoStep_PCA, MixtureFA, MixturePCA
- 2 model types: Mixture (static) vs Dynamic_v2 (with 3 orthogonal axes)
- Metrics: ARI (clustering accuracy), VAF (reconstruction quality), BIC, SSE
- Variable parameters: N, K, r, cluster_sep_spatial, cluster_sep_temporal, cluster_sep_stability

For parallel processing (cores > 1), the function uses parLapplyLB for load balancing. If the pbapply package is installed, a progress bar will be displayed.

Value

Invisibly returns a list containing:

- raw_results: Data frame with raw results from all runs
- aggregated_results: Data frame with results aggregated across seeds
- summary_file: Path to the summary CSV file
- win_rates_file: Path to the win rates CSV file
- output_dir: Directory where results were saved

Examples

```
## Not run:
# Run test mode with 2 cores
results <- run_param_sweep(test = TRUE, cores = 2)

# Run full sweep with 8 cores
results <- run_param_sweep(test = FALSE, cores = 8)

# Run with custom output directory
results <- run_param_sweep(test = TRUE, cores = 2, output_dir = "my_results")

# Run with custom parameter grid
library(dplyr)
custom_grid <- expand_grid(
  N = c(20, 50),
  K = c(2, 3),
  r = 2,
  M = 8,
  T_each = 200,
  cluster_sep_spatial = c(0.3, 0.6),
  cluster_sep_temporal = 0.6,
  cluster_sep_stability = 1.0,
  seed = 1:3,
  stringsAsFactors = FALSE
)
custom_grid$run_id <- seq_len(nrow(custom_grid))

results <- run_param_sweep(
  param_grid = custom_grid,
  cores = 8,
  output_dir = "inst/paper/results_custom"
)

# Access results directly
head(results$raw_results)
head(results$aggregated_results)

## End(Not run)
```

run_param_sweep_baselines

Run Parameter Sweep for Baseline Method Comparisons (v2)

Description

Orchestrates a parameter sweep experiment to evaluate baseline method performance (SingleFA, SinglePCA, TwoStep_FA, TwoStep_PCA, MixtureFA, MixturePCA) using the v2 dynamic synergy simulator with three orthogonal control axes. This function focuses on method-level benchmarking rather than model comparison.

Usage

```
run_param_sweep_baselines(
  test = FALSE,
  grid_type = NULL,
  cores = 4,
  output_dir = NULL,
  param_grid = NULL,
  verbose = TRUE,
  align_basis = TRUE,
  refine_assignments = TRUE,
  parallel_strategy = c("grid", "sequential", "pbs", "future"),
  pbs_config = list(),
  pbs_submit = FALSE,
  pbs_wait = FALSE,
  future_backend = "multisession",
  future_resources = list()
)
```

Arguments

test	Logical. If TRUE, uses a smaller test grid for quick validation. Default is FALSE (full parameter grid).
grid_type	Character. Parameter grid type: "test", "paper_core", or "paper_stress". If NULL (default), uses the test parameter to determine the grid. When specified, overrides the test parameter.
cores	Integer. Number of CPU cores to use for parallel processing. Default is 4. Set to 1 for sequential processing.
output_dir	Character. Directory to save results. Default is "inst/paper/results_baselines_v2" if it exists (repo use), otherwise "results_baselines_v2" in the current working directory.
param_grid	Optional. Custom parameter grid data frame. If NULL, uses the built-in full or test grid based on the test parameter. Required columns: N, K, r, M, T_each, cluster_sep_spatial, cluster_sep_temporal, cluster_sep_stability, seed, run_id.
verbose	Logical. If TRUE, prints progress information. Default is TRUE.
align_basis	Logical. If TRUE (default), applies Hungarian algorithm alignment in Step 1.5 of TwoStep methods to ensure corresponding components across subjects are in the same order before clustering. Set to FALSE to disable alignment and compare performance with/without this step in the parameter sweep.
refine_assignments	Logical. If TRUE (default), performs a single reassignment pass after fitting cluster-specific models in Step 4 of TwoStep methods. Each subject is reassigned to the cluster that minimizes reconstruction error. Set to FALSE to disable reassignment for ablation studies.
parallel_strategy	Character. Parallelization strategy: "grid": Local multicore parallelization using parLapplyLB

	• "sequential": No parallelization (single core) • "pbs": PBS/OpenPBS array job submission for HPC clusters • "future": Use future framework (supports local and HPC backends) Default is "grid".
<code>pbs_config</code>	List. PBS configuration options (only used when <code>parallel_strategy = "pbs"</code>). See submit_pbs_sweep for details.
<code>pbs_submit</code>	Logical. If TRUE and <code>parallel_strategy = "pbs"</code> , submit the job to PBS. If FALSE, only generate scripts (dry run). Default is FALSE.
<code>pbs_wait</code>	Logical. If TRUE, wait for PBS job completion and aggregate results. Default is FALSE.
<code>future_backend</code>	Character. Backend for future parallelization (only used when <code>parallel_strategy = "future"</code>): • "multisession": Local parallel R sessions (default) • "multicore": Local forked processes (Unix only) • "batchtools_torque": PBS/TORQUE scheduler • "batchtools_slurm": SLURM scheduler See setup_future_backend for all options.
<code>future_resources</code>	List. HPC resource specifications for future.batchtools backends (e.g., <code>list(walltime = "04:00:00", memory = "4gb")</code>).

Details

This function complements [run_param_sweep](#) by focusing on method-level benchmarking rather than model comparison. It evaluates:

- 6 methods: SingleFA, SinglePCA, TwoStep_FA, TwoStep_PCA, MixtureFA, MixturePCA
- 1 model type: Dynamic_v2 (with 3 orthogonal axes)
- Metrics: ARI (clustering accuracy), VAF (reconstruction quality), BIC, SSE
- Variable parameters: N, K, r, cluster_sep_spatial, cluster_sep_temporal, cluster_sep_stability

For parallel processing (`cores > 1`), the function uses `parLapplyLB` for load balancing. If the `pbapply` package is installed, a progress bar will be displayed.

Value

Invisibly returns a list containing:

- `raw_results`: Data frame with raw results from all runs
- `aggregated_results`: Data frame with results aggregated across seeds
- `summary_file`: Path to the summary CSV file
- `win_rates_file`: Path to the win rates CSV file
- `output_dir`: Directory where results were saved

PBS Mode

When `parallel_strategy = "pbs"`, this function generates PBS array job scripts and optionally submits them. Each array task runs one parameter combination. Results are saved as individual RDS files and can be aggregated after job completion using [aggregate_partial_results](#).

Future Mode

When `parallel_strategy = "future"`, this function uses the future framework for flexible parallelization. Local backends (multisession, multicore) work out of the box. HPC backends (batchtools_torque, batchtools_slurm) require the future.batchtools package and appropriate cluster configuration.

See Also

[compare_baselines](#) for single-run baseline comparison, [run_param_sweep](#) for model comparison parameter sweep, [plot_sweep_summary](#) for visualization

Examples

```
## Not run:
# Run test mode with 2 cores
results <- run_param_sweep_baselines(test = TRUE, cores = 2)

# Run full sweep with 8 cores
results <- run_param_sweep_baselines(test = FALSE, cores = 8)

# Run with custom output directory
results <- run_param_sweep_baselines(
  test = TRUE,
  cores = 2,
  output_dir = "my_baselines_results"
)

# Use future with local multisession backend
results <- run_param_sweep_baselines(
  test = TRUE,
  parallel_strategy = "future",
  future_backend = "multisession",
  cores = 4
)

# Use PBS for HPC cluster
results <- run_param_sweep_baselines(
  test = FALSE,
  parallel_strategy = "pbs",
  pbs_config = list(queue = "batch", walltime = "04:00:00", mem = "4gb"),
  pbs_submit = TRUE,
  pbs_wait = TRUE
)

# Run with custom parameter grid
```

```

library(dplyr)
custom_grid <- expand.grid(
  N = c(20, 50),
  K = c(2, 3),
  r = 2,
  M = 8,
  T_each = 200,
  cluster_sep_spatial = c(0.3, 0.6),
  cluster_sep_temporal = 0.6,
  cluster_sep_stability = 1.0,
  seed = 1:3,
  stringsAsFactors = FALSE
)
custom_grid$run_id <- seq_len(nrow(custom_grid))

results <- run_param_sweep_baselines(
  param_grid = custom_grid,
  cores = 8,
  output_dir = "inst/paper/results_baselines_custom"
)

# Access results directly
head(results$raw_results)
head(results$aggregated_results)

## End(Not run)

```

run_param_sweep_modelsel

Run Parameter Sweep with Model Selection

Description

Orchestrates a comprehensive parameter sweep experiment that includes model selection (K, r) for each parameter combination. This function generates v2 dynamic synergy data, performs model selection using specified strategies, and evaluates performance metrics.

Usage

```

run_param_sweep_modelsel(
  test = FALSE,
  grid_type = NULL,
  mode = c("modelsel", "oracle_r", "oracle_Kr"),
  cores = 4,
  output_dir = NULL,
  param_grid = NULL,
  K_candidates = 1:5,
  r_candidates = 1:5,

```

```

strategies = c("grid", "twostage"),
methods = c("MFA", "MPCA"),
criterion = c("ICL", "BIC"),
icl_temperature = 2,
parallel_strategy = c("grid", "sequential", "pbs", "future"),
pbs_config = list(),
pbs_submit = FALSE,
pbs_wait = FALSE,
future_backend = "multisession",
future_resources = list(),
max_iter = 50,
nIterFA = 5,
nIterPCA = 5,
n_init = 1,
tol = 0.001,
use_kmeans_init = TRUE,
n_threads = 1,
save_models = FALSE,
save_selection_summaries = TRUE,
verbose = TRUE
)

```

Arguments

test	Logical. If TRUE, uses a smaller test grid for quick validation. Default is FALSE (full parameter grid).
grid_type	Character. Parameter grid type: "test", "paper_core", or "paper_stress". If NULL (default), uses the test parameter to determine the grid. When specified, overrides the test parameter.
mode	Character. Selection mode controlling the search behavior: • "modelsel": Normal model selection (K,r grid search) • "oracle_r": Use true r, only select K • "oracle_Kr": Use true K and r (no selection) Default is "modelsel".
cores	Integer. Number of CPU cores for parallel processing. Default is 4. Set to 1 for sequential processing.
output_dir	Character. Directory to save results. Default is "inst/paper/results_sweep_modelsel" if it exists, otherwise "results_sweep_modelsel" in the current working directory.
param_grid	Optional. Custom parameter grid data frame. If NULL, uses the built-in full or test grid based on the test parameter.
K_candidates	Integer vector. Candidate K values for model selection. Default is 1:5.
r_candidates	Integer vector. Candidate r values for model selection. Default is 1:5.
strategies	Character vector. Model selection strategies to evaluate. Options: "grid", "twostage". Default is c("grid", "twostage").
methods	Character vector. Model types to evaluate. Options: "MFA", "MPCA". Default is c("MFA", "MPCA").

criterion	Character. Model selection criterion: "ICL" or "BIC". Default is "ICL".
icl_temperature	Numeric. Temperature parameter for ICL entropy scaling. Higher values favor more clusters. Default is 2.0.
parallel_strategy	Character. Parallelization strategy: • "grid": Local multicore parallelization using mclapply • "sequential": No parallelization (single core) • "pbs": PBS/OpenPBS array job submission for HPC clusters • "future": Use future framework (supports local and HPC backends) Default is "grid".
pbs_config	List. PBS configuration options (only used when parallel_strategy = "pbs"). See submit_pbs_sweep for details.
pbs_submit	Logical. If TRUE and parallel_strategy = "pbs", submit the job to PBS. If FALSE, only generate scripts (dry run). Default is FALSE.
pbs_wait	Logical. If TRUE, wait for PBS job completion and aggregate results. Default is FALSE.
future_backend	Character. Backend for future parallelization (only used when parallel_strategy = "future"): • "multisession": Local parallel R sessions (default) • "multicore": Local forked processes (Unix only) • "batchtools_torque": PBS/TORQUE scheduler • "batchtools_slurm": SLURM scheduler See setup_future_backend for all options.
future_resources	List. HPC resource specifications for future.batchtools backends (e.g., list(walltime = "04:00:00", memory = "4gb")).
max_iter	Maximum EM iterations. Default is 50.
nIterFA	Sub-iterations for FA update. Default is 5.
nIterPCA	Sub-iterations for PCA update. Default is 5.
n_init	Number of random initializations. Default is 1.
tol	Convergence tolerance. Default is 1e-3.
use_kmeans_init	Whether to use k-means initialization. Default is TRUE.
n_threads	Number of OpenMP threads. Default is 1.
save_models	Logical. If TRUE, save fitted models. Default is FALSE.
save_selection_summaries	Logical. If TRUE, save BIC summaries for each run. Default is TRUE.
verbose	Logical. If TRUE, print progress information. Default is TRUE.

Details

The parameter sweep evaluates model selection performance across:

- Methods: MFA, MPCA
- Strategies: grid (full BIC grid search), twostage (K first, then r)
- Metrics: ARI (clustering accuracy), VAF (reconstruction quality), BIC, selection accuracy (K_correct, r_correct, Kr_correct)
- Variable parameters: N, K, r, cluster_sep_spatial, cluster_sep_temporal, cluster_sep_stability

Value

Invisibly returns a list containing:

- raw_results: Data frame with raw results from all runs
- aggregated_results: Data frame with results aggregated across seeds
- win_rates: Data frame with win rates by method/strategy
- output_dir: Directory where results were saved

Output Schema

The raw_results data frame contains the following columns:

- run_id, seed: Identifiers
- N, K_true, r_true, M, T_each: True parameters
- cluster_sep_spatial, cluster_sep_temporal, cluster_sep_stability: Separation parameters
- method: "MFA" or "MPCA"
- selection_strategy: "grid" or "twostage"
- mode: Selection mode ("modelsel", "oracle_r", or "oracle_Kr")
- r_mode: Always "global" (for future cluster-wise r extension)
- K_hat, r_hat: Estimated parameters
- K_correct, r_correct, Kr_correct: Selection accuracy
- ARI, VAF, logLik, BIC: Performance metrics
- subspace_similarity: Subspace recovery score (mean across clusters)
- runtime_sec: Runtime in seconds
- error_msg: Error message (empty string if successful)

Examples

```
## Not run:
# Run test mode with 2 cores
results <- run_param_sweep_modelsel(test = TRUE, cores = 2)

# Run full sweep with 8 cores
results <- run_param_sweep_modelsel(test = FALSE, cores = 8)

# Run with custom strategies
results <- run_param_sweep_modelsel(
  test = TRUE, cores = 4,
  strategies = c("twostage"),
  methods = c("MFA")
)

## End(Not run)
```

run_single_task	<i>Run a Single Task from Parameter Grid</i>
-----------------	--

Description

Executes a single parameter combination from the parameter grid. This function is called by PBS array job workers.

Usage

```
run_single_task(
  task_id,
  param_grid_file,
  output_dir,
  sweep_type = c("modelsel", "baselines"),
  methods = c("MFA", "MPCA"),
  strategies = c("grid", "twostage"),
  K_candidates = 1:5,
  r_candidates = 1:5,
  mode = c("modelsel", "oracle_r", "oracle_Kr"),
  criterion = c("ICL", "BIC"),
  icl_temperature = 2,
  max_iter = 50,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  align_basis = TRUE,
  refine_assignments = TRUE,
  verbose = TRUE
)
```

Arguments

task_id	Integer. The task ID corresponding to run_id in the parameter grid.
param_grid_file	Character. Path to the RDS file containing the parameter grid.
output_dir	Character. Directory to save results.
sweep_type	Character. Type of sweep: "modelsel" or "baselines".
methods	Character vector. Methods to evaluate (e.g., c("MFA", "MPCA")).
strategies	Character vector. Selection strategies (e.g., c("grid", "twostage")).
K_candidates	Integer vector. Candidate K values for model selection.
r_candidates	Integer vector. Candidate r values for model selection.
mode	Character. Selection mode: "modelsel", "oracle_r", or "oracle_Kr".
criterion	Character. Model selection criterion: "ICL" or "BIC".
icl_temperature	Numeric. Temperature parameter for ICL entropy scaling.
max_iter	Integer. Maximum EM iterations.
tol	Numeric. Convergence tolerance.
n_init	Integer. Number of random initializations.
use_kmeans_init	Logical. Whether to use k-means initialization.
align_basis	Logical. For baselines sweep: whether to align basis vectors.
refine_assignments	Logical. For baselines sweep: whether to refine cluster assignments.
verbose	Logical. Whether to print progress.

Details

This function reads the parameter grid from a file, extracts the row corresponding to the given task_id, runs the computation, and saves the result to a separate file. This enables PBS array jobs where each task processes one parameter combination independently.

Value

Invisibly returns TRUE on success.

Examples

```
## Not run:
# Called from PBS script:
# Rscript -e "synergyMixR::run_single_task(
#   task_id = as.integer(Sys.getenv('PBS_ARRAY_INDEX')),
#   param_grid_file = 'results/param_grid.rds',
#   output_dir = 'results'
# )"

## End(Not run)
```

run_sweep_future	<i>Run Parameter Sweep Using Future</i>
------------------	---

Description

Executes a parameter sweep using the future framework for parallelization. This works with any future backend (local, PBS, SLURM, etc.).

Usage

```
run_sweep_future(
  param_grid,
  run_func,
  ...,
  future_backend = "multisession",
  workers = NULL,
  resources = list(),
  chunk_size = NULL,
  verbose = TRUE
)
```

Arguments

param_grid	Data frame. Parameter grid to sweep.
run_func	Function. Function to run for each parameter combination. Should accept a single row of param_grid and return a data frame.
...	Additional arguments passed to run_func.
future_backend	Character. Backend to use (see setup_future_backend).
workers	Integer. Number of workers for local backends.
resources	List. HPC resources for batchtools backends.
chunk_size	Integer. Number of tasks per future (for load balancing).
verbose	Logical. Print progress information.

Value

Data frame with combined results from all parameter combinations.

Examples

```
## Not run:
# Define a simple run function
my_run_func <- function(p) {
  Sys.sleep(1) # Simulate work
  data.frame(run_id = p$run_id, result = p$x^2)
}
```

```

# Create parameter grid
param_grid <- data.frame(run_id = 1:10, x = 1:10)

# Run with local parallelization
results <- run_sweep_future(
  param_grid, my_run_func,
  future_backend = "multisession",
  workers = 2
)

# Run with PBS
results <- run_sweep_future(
  param_grid, my_run_func,
  future_backend = "batchtools_torque",
  resources = list(walltime = "01:00:00")
)

## End(Not run)

```

scale_color_mixsynergy

Get Unified Color Scale for Discrete Categories

Description

Returns a consistent color scale for discrete categories using viridis.

Usage

```
scale_color_mixsynergy(option = "viridis", ...)
```

Arguments

option	Viridis color option: "viridis" (default), "magma", "plasma", "inferno", "cividis", "rocket", "mako", "turbo"
...	Additional arguments passed to scale_color_viridis_d()

Value

A ggplot2 scale object

Examples

```

## Not run:
library(ggplot2)
ggplot(data, aes(x, y, color = cluster)) +
  geom_point() +
  scale_color_mixsynergy()

## End(Not run)

```

scale_fill_mixsynergy *Get Unified Viridis Color Scale for Continuous Values*

Description

Returns a consistent viridis color scale for continuous fill aesthetics. Uses the "viridis" option by default for colorblind-friendly visualization.

Usage

```
scale_fill_mixsynergy(option = "viridis", direction = 1, ...)
```

Arguments

option	Viridis color option: "viridis" (default), "magma", "plasma", "inferno", "cividis", "rocket", "mako", "turbo"
direction	Direction of color scale: 1 (default) or -1 (reversed)
...	Additional arguments passed to scale_fill_viridis_c()

Value

A ggplot2 scale object

Examples

```
## Not run:  
library(ggplot2)  
ggplot(data, aes(x, y, fill = value)) +  
  geom_tile() +  
  scale_fill_mixsynergy()  
  
## End(Not run)
```

scale_fill_mixsynergy_discrete *Get Unified Fill Scale for Discrete Categories*

Description

Returns a consistent fill scale for discrete categories using viridis. Use this for bar plots, boxplots, and other geoms with discrete fill aesthetics.

Usage

```
scale_fill_mixsynergy_discrete(option = "viridis", ...)
```

Arguments

option	Viridis color option: "viridis" (default), "magma", "plasma", "inferno", "cividis", "rocket", "mako", "turbo"
...	Additional arguments passed to scale_fill_viridis_d()

Value

A ggplot2 scale object

Examples

```
## Not run:
library(ggplot2)
ggplot(data, aes(x, y, fill = cluster)) +
  geom_col() +
  scale_fill_mixsynergy_discrete()

## End(Not run)
```

scale_fill_mixsynergy_diverging
<i>Get Unified Viridis Color Scale for Diverging Values</i>

Description

Returns a consistent color scale for diverging data (e.g., correlations, loadings). Uses a diverging palette centered at zero.

Usage

```
scale_fill_mixsynergy_diverging(
  option = "RdBu",
  midpoint = 0,
  limits = NULL,
  ...
)
```

Arguments

option	Viridis color option for diverging scale (default: "RdBu")
midpoint	Midpoint value for diverging scale (default: 0)
limits	Limits for the color scale (default: NULL for automatic)
...	Additional arguments passed to scale_fill_gradient2()

Value

A ggplot2 scale object

Examples

```
## Not run:
library(ggplot2)
ggplot(data, aes(x, y, fill = correlation)) +
  geom_tile() +
  scale_fill_mixed(synergy_diverging())

## End(Not run)
```

```
select_K_bootstrap_stability
```

Select K Using Bootstrap Stability Selection

Description

This function implements bootstrap stability selection for choosing the optimal number of clusters K. It assesses the stability of cluster assignments across bootstrap resamples and selects K that produces the most stable clustering.

Usage

```
select_K_bootstrap_stability(
  list_of_data,
  Kvec = 2:5,
  r = 2,
  B = 50,
  subsample_ratio = 0.8,
  max_iter = 50,
  n_init = 1,
  model_type = c("MFA", "MPCA"),
  verbose = TRUE
)
```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>Kvec</code>	Vector of candidate K values to test.
<code>r</code>	Number of factors (fixed).
<code>B</code>	Number of bootstrap iterations (default: 50).
<code>subsample_ratio</code>	Proportion of subjects to sample (default: 0.8).
<code>max_iter</code>	Maximum EM iterations per fit.
<code>n_init</code>	Number of random initializations per fit.
<code>model_type</code>	Either "MFA" or "MPCA" (default: "MFA").
<code>verbose</code>	Print progress messages (default: TRUE).

Details

The stability selection procedure: 1. For each candidate K, perform B bootstrap iterations 2. In each iteration, subsample subjects and fit the model 3. Compute pairwise co-clustering matrix (1 if subjects i,j in same cluster) 4. Average co-clustering matrices across bootstrap samples 5. Compute stability score as mean of (max probability per subject - 1/K) 6. Select K with highest stability score

Higher stability indicates more consistent cluster assignments across resamples.

Value

A list containing:

best_K The selected optimal K based on stability

stability_scores Stability score for each K

stability_df Data frame with K and stability scores

References

Ben-Hur, A., Elisseeff, A., & Guyon, I. (2001). A stability based method for discovering structure in clustered data. Pacific Symposium on Biocomputing.

select_K_crossvalidation

Select K Using Cross-Validation

Description

This function implements K-fold cross-validation for choosing the optimal number of clusters K. It evaluates model generalization by computing log-likelihood on held-out subjects.

Usage

```
select_K_crossvalidation(
  list_of_data,
  Kvec = 2:5,
  r = 2,
  nfolds = 5,
  max_iter = 50,
  n_init = 1,
  model_type = c("MFA", "MPCA"),
  verbose = TRUE
)
```

Arguments

list_of_data	A list of data matrices, each (T _i x M).
Kvec	Vector of candidate K values to test.
r	Number of factors (fixed).
nfolds	Number of cross-validation folds (default: 5).
max_iter	Maximum EM iterations per fit.
n_init	Number of random initializations per fit.
model_type	Either "MFA" or "MPCA" (default: "MFA").
verbose	Print progress messages (default: TRUE).

Details

The cross-validation procedure: 1. Split subjects into nfolds groups 2. For each K and each fold: - Train model on (nfolds-1) folds - Compute log-likelihood on held-out fold 3. Average log-likelihoods across folds for each K 4. Select K with highest average CV log-likelihood
Higher CV log-likelihood indicates better generalization.

Value

A list containing:
best_K The selected optimal K based on CV log-likelihood
cv_scores Mean CV log-likelihood for each K
cv_se Standard error of CV log-likelihood for each K
cv_df Data frame with K, mean CV score, and SE

select_K_r	<i>Unified Model Selection for K and r</i>
------------	--

Description

This function provides a unified interface for selecting optimal K (number of clusters) and r (factor dimension) for mixture models (MFA or MPCA). It supports multiple selection strategies including grid search and two-stage selection.

Usage

```
select_K_r(  
  list_of_data,  
  method = c("MFA", "MPCA"),  
  strategy = c("grid", "twostage"),  
  Kvec = 1:5,  
  rvec = 1:5,  
  r_fixed = 2,
```

```

criterion = c("ICL", "BIC"),
icl_max_iter = NULL,
icl_temperature = 2,
max_iter = 50,
nIterFA = 5,
nIterPCA = 5,
tol = 0.001,
mpca_method = "EM",
n_init = 1,
use_kmeans_init = TRUE,
subject_rdim_for_kmeans = 2,
mc_cores_grid = 1,
mc_cores = 1,
n_threads = 1,
verbose = TRUE
)

```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>method</code>	Model type: "MFA" or "MPCA".
<code>strategy</code>	Selection strategy: "grid" for full grid search, "twostage" for two-stage sequential selection (K first, then r).
<code>Kvec</code>	Integer vector of candidate K values. Default is 1:5.
<code>rvec</code>	Integer vector of candidate r values. Default is 1:5.
<code>r_fixed</code>	Fixed r value for Stage 1 of twostage selection. Default is 2.
<code>criterion</code>	Model selection criterion: "ICL" (Integrated Completed Likelihood) or "BIC" (Bayesian Information Criterion). Default is "ICL".
<code>icl_max_iter</code>	Maximum iterations for ICL entropy calculation. Default is NULL (uses package default).
<code>icl_temperature</code>	Temperature parameter for ICL entropy scaling. Higher values favor more clusters. Default is 2.0.
<code>max_iter</code>	Maximum EM iterations. Default is 50.
<code>nIterFA</code>	Sub-iterations for FA update (MFA only). Default is 5.
<code>nIterPCA</code>	Sub-iterations for PCA update (MPCA only). Default is 5.
<code>tol</code>	Convergence tolerance. Default is 1e-3.
<code>mpca_method</code>	Method for MPCA: "EM" or "closed_form". Default is "EM".
<code>n_init</code>	Number of random initializations. Default is 1.
<code>use_kmeans_init</code>	Whether to use k-means initialization. Default is TRUE.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features. Default is 2.
<code>mc_cores_grid</code>	Number of cores for grid-level parallelization. Default is 1.

mc_cores	Number of cores for initialization-level parallelization. Default is 1.
n_threads	Number of OpenMP threads. Default is 1.
verbose	Logical; if TRUE, print progress messages. Default is TRUE.

Details

This function wraps the existing model selection functions to provide a unified interface for parameter sweep experiments. It handles:

- Parallel safety: Prevents nested parallelization by forcing `n_threads=1` when `mc_cores_grid > 1`
- Windows compatibility: Falls back to sequential execution on Windows
- Consistent output format: Returns the same structure regardless of method or strategy

Strategy: "grid" Performs full grid search over all (K, r) combinations using BIC for selection. This is thorough but computationally expensive.

Strategy: "twostage" Two-stage sequential selection:

1. Stage 1: Fix r at `r_fixed`, search for optimal K using BIC
2. Stage 2: Fix K at optimal value, search for optimal r using BIC

This is more stable and faster than full grid search.

Value

A list with unified output structure:

`best_K` Optimal number of clusters (scalar).

`best_r` Optimal factor dimension (scalar).

`best_model` Fitted model object with optimal (K, r).

`summary_df` Data frame with selection summary. For grid strategy, contains all (K, r) combinations with BIC. For twostage, contains stage1 and stage2 summaries.

`meta` List with metadata: method, strategy, Kvec, rvec, etc.

See Also

[select_optimal_K_r_mfa](#), [select_optimal_K_r_mpca](#), [select_optimal_K_r_mfa_twostage](#), [select_optimal_K_r_mpca_twostage](#)

Examples

```
## Not run:
# Grid search for MFA
result <- select_K_r(
  list_of_data, method = "MFA", strategy = "grid",
  Kvec = 2:4, rvec = 2:4
)
print(result$best_K)
print(result$best_r)
```

```
# Two-stage selection for MPCA
result <- select_K_r(
  list_of_data, method = "MPCA", strategy = "twostage",
  Kvec = 2:4, rvec = 2:4, r_fixed = 2
)

## End(Not run)
```

```
select_optimal_K_r_mfa
```

Search for the Best (K, r) for MFA via Grid Search (Using mfa_em_fit)

Description

This function tries all combinations of K in Kvec and r in rvec, fits a Mixture Factor Analysis (MFA) model using mfa_em_fit for each combination, computes BIC, and returns the best model by BIC.

Usage

```
select_optimal_K_r_mfa(
  list_of_data,
  Kvec = 1:5,
  rvec = 1:5,
  max_iter = 50,
  nIterFA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores_grid = 1,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

list_of_data	A list of data matrices, each (T _i x M).
Kvec	An integer vector of candidate K values.
rvec	An integer vector of candidate r values.
max_iter	Maximum EM iterations.
nIterFA	Sub-iterations within the factor-analyzer update in C++.
tol	Convergence tolerance.
n_init	Number of random initializations to try inside mfa_em_fit.

use_kmeans_init	Logical; whether to also try a k-means-based init.
subject_rdim_for_kmeans	Dimension for PCA-based features if use_kmeans_init=TRUE.
mc_cores_grid	Number of CPU cores for grid-level parallelization (default: 1).
mc_cores	Number of CPU cores for initialization-level parallelization within each (K, r) (default: 1).
n_threads	Number of OpenMP threads for EM-level parallelization (default: 1 to avoid nested parallelism).

Details

- Calls `mfa_em_fit(..., n_init, use_kmeans_init, ...)` for each (K, r). - After fitting, we compute log-likelihood by summing over all data (you can use the built-in `logLik` from the fit, or re-compute).
 - Then compute $BIC = -2 \cdot \log Lik + p \cdot \log(N_{total_rows})$. Here $p = K \cdot (M \cdot r + 2 \cdot M) + (K - 1)$ by default (diagonal noise).

Value

A list with:

summary	A data.frame with columns K, r, logLik, BIC.
best_model_info	A list with (K, r, model, logLik, BIC) for the best BIC.
all_models	A list of all fits, each (K, r, model, logLik, BIC).

select_optimal_K_r_mfa_twostage

Two-Stage Model Selection for MFA: Select K First, Then r

Description

This function implements a more stable two-stage approach to model selection: Stage 1: Fix r (e.g., r=2) and search for optimal K using BIC or ICL Stage 2: Fix K at the optimal value and search for optimal r using BIC

Usage

```
select_optimal_K_r_mfa_twostage(
  list_of_data,
  Kvec = 2:5,
  rvec = 1:5,
  r_fixed = 2,
  criterion = c("ICL", "BIC"),
  max_iter = 50,
  icl_max_iter = NULL,
```

```

    icl_temperature = 2,
    nIterFA = 5,
    tol = 0.001,
    n_init = 1,
    use_kmeans_init = TRUE,
    subject_rdim_for_kmeans = 2,
    mc_cores = 1,
    n_threads = 1
)

```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>Kvec</code>	An integer vector of candidate K values for Stage 1.
<code>rvec</code>	An integer vector of candidate r values for Stage 2.
<code>r_fixed</code>	Fixed r value to use in Stage 1 (default: 2).
<code>criterion</code>	Model selection criterion for K selection in Stage 1. Either "BIC" (default) or "ICL". ICL adds an entropy penalty that discourages uncertain cluster assignments, making it more conservative for K selection. Stage 2 (r selection) always uses BIC.
<code>max_iter</code>	Maximum EM iterations.
<code>icl_max_iter</code>	Maximum EM iterations for ICL entropy calculation (default: 10). Only used when <code>criterion="ICL"</code> . This limits iterations to maintain soft cluster assignments (non-zero entropy), making ICL effective. If NULL, defaults to 10. Uses random initialization (not K-means) internally.
<code>icl_temperature</code>	Temperature parameter for ICL entropy scaling (default: 2.0). Higher values favor more clusters by scaling down the entropy penalty.
<code>nIterFA</code>	Sub-iterations within the factor-analyzer update in C++.
<code>tol</code>	Convergence tolerance.
<code>n_init</code>	Number of random initializations to try.
<code>use_kmeans_init</code>	Logical; whether to also try a k-means-based init.
<code>subject_rdim_for_kmeans</code>	Dimension for PCA-based features if <code>use_kmeans_init=TRUE</code> .
<code>mc_cores</code>	Number of CPU cores for initialization-level parallelization (default: 1).
<code>n_threads</code>	Number of OpenMP threads for EM-level parallelization (default: 1).

Details

This approach is more stable than simultaneous (K, r) grid search because K and r have strong interactions in mixture models.

This two-stage approach is recommended in the literature for mixture models because it avoids the instability of simultaneous (K, r) optimization. The default `r_fixed=2` is a reasonable starting point for most applications.

When criterion="ICL", the Integrated Complete-data Likelihood is used for K selection. ICL adds an entropy penalty to BIC that discourages models with uncertain cluster assignments. This makes ICL more conservative than BIC and can help avoid overestimating K.

Value

A list with:

stage1_summary	Data frame with K search results (r fixed). Contains BIC and optionally ICL if criterion="ICL".
stage2_summary	Data frame with r search results (K fixed).
best_K	Optimal K from Stage 1.
best_r	Optimal r from Stage 2.
best_model_info	Final model with optimal (K, r).
best_model	The fitted model object.
criterion	The criterion used for K selection.

References

Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7), 719-725.

select_optimal_K_r_mPCA

Search for the Best (K, r) for Mixture PCA via Grid Search

Description

This function tries all combinations of K in Kvec and r in rvec, fits a Mixture PCA (PPCA) model using mixture_pca_em_fit for each combination, computes BIC, and returns the best model by BIC.

Usage

```
select_optimal_K_r_mPCA(
  list_of_data,
  Kvec = 1:5,
  rvec = 1:5,
  max_iter = 50,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
```

```

mc_cores_grid = 1,
mc_cores = 1,
n_threads = 1
)

```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>Kvec</code>	An integer vector of candidate K values.
<code>rvec</code>	An integer vector of candidate r values.
<code>max_iter</code>	Maximum EM iterations.
<code>nIterPCA</code>	Sub-iterations within the PCA/PPCA update in C++.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form" (passed to <code>mixture_pca_em_fit</code>).
<code>n_init</code>	Number of random initializations to try inside <code>mixture_pca_em_fit</code> .
<code>use_kmeans_init</code>	Logical; whether to also try a k-means-based init.
<code>subject_rdim_for_kmeans</code>	Dimension for PCA-based features if <code>use_kmeans_init=TRUE</code> .
<code>mc_cores_grid</code>	Number of CPU cores for grid-level parallelization (default: 1).
<code>mc_cores</code>	Number of CPU cores for initialization-level parallelization within each (K, r) (default: 1).
<code>n_threads</code>	Number of OpenMP threads for EM-level parallelization (default: 1 to avoid nested parallelism).

Details

- Calls `mixture_pca_em_fit(..., n_init, use_kmeans_init, ...)` for each (K, r). - Uses `fit$logLik` as the final log-likelihood. - BIC formula can be adjusted if needed. By default, we might do $p = K \cdot (M \cdot r + r + M) + (K-1)$, for example, if we consider each cluster having: - W is $M \times r$ (or equivalently $P \times D$) - μ is length M - σ^2 is 1 scalar - mixing proportions ($K-1$)

Value

A list with:

<code>summary</code>	A data.frame with columns $K, r, \log\text{Lik}, \text{BIC}$.
<code>best_model_info</code>	A list with ($K, r, \text{model}, \log\text{Lik}, \text{BIC}$) for the best BIC.
<code>all_models</code>	A list of all fits, each ($K, r, \text{model}, \log\text{Lik}, \text{BIC}$).

select_optimal_K_r_mPCA_twostage

Two-Stage Model Selection for MPCA: Select K First, Then r

Description

This function implements a more stable two-stage approach to model selection: Stage 1: Fix r (e.g., $r=2$) and search for optimal K using BIC or ICL Stage 2: Fix K at the optimal value and search for optimal r using BIC

Usage

```
select_optimal_K_r_mPCA_twostage(
  list_of_data,
  Kvec = 2:5,
  rvec = 1:5,
  r_fixed = 2,
  criterion = c("ICL", "BIC"),
  max_iter = 50,
  icl_max_iter = NULL,
  icl_temperature = 2,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>Kvec</code>	An integer vector of candidate K values for Stage 1.
<code>rvec</code>	An integer vector of candidate r values for Stage 2.
<code>r_fixed</code>	Fixed r value to use in Stage 1 (default: 2).
<code>criterion</code>	Model selection criterion for K selection in Stage 1. Either "BIC" (default) or "ICL". ICL adds an entropy penalty that discourages uncertain cluster assignments, making it more conservative for K selection. Stage 2 (r selection) always uses BIC.
<code>max_iter</code>	Maximum EM iterations.
<code>icl_max_iter</code>	Maximum EM iterations for ICL entropy calculation (default: 10). Only used when <code>criterion="ICL"</code> . This limits iterations to maintain soft cluster assignments (non-zero entropy), making ICL effective. If NULL, defaults to 10. Uses random initialization (not K-means) internally.

<code>icl_temperature</code>	Temperature parameter for ICL entropy scaling (default: 2.0). Higher values favor more clusters by scaling down the entropy penalty.
<code>nIterPCA</code>	Sub-iterations within the PCA/PPCA update in C++.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form" (passed to <code>mixture_pca_em_fit</code>).
<code>n_init</code>	Number of random initializations to try.
<code>use_kmeans_init</code>	Logical; whether to also try a k-means-based init.
<code>subject_rdim_for_kmeans</code>	Dimension for PCA-based features if <code>use_kmeans_init=TRUE</code> .
<code>mc_cores</code>	Number of CPU cores for initialization-level parallelization (default: 1).
<code>n_threads</code>	Number of OpenMP threads for EM-level parallelization (default: 1).

Details

This approach is more stable than simultaneous (K, r) grid search because K and r have strong interactions in mixture models.

This two-stage approach is recommended in the literature for mixture models because it avoids the instability of simultaneous (K, r) optimization. The default `r_fixed=2` is a reasonable starting point for most applications.

When `criterion="ICL"`, the Integrated Complete-data Likelihood is used for K selection. ICL adds an entropy penalty to BIC that discourages models with uncertain cluster assignments. This makes ICL more conservative than BIC and can help avoid overestimating K.

Value

A list with:

<code>stage1_summary</code>	Data frame with K search results (r fixed). Contains BIC and optionally ICL if <code>criterion="ICL"</code> .
<code>stage2_summary</code>	Data frame with r search results (K fixed).
<code>best_K</code>	Optimal K from Stage 1.
<code>best_r</code>	Optimal r from Stage 2.
<code>best_model_info</code>	Final model with optimal (K, r).
<code>best_model</code>	The fitted model object.
<code>criterion</code>	The criterion used for K selection.

References

Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7), 719-725.

select_r_by_smoothness_mfa

Select Optimal r by Smoothness Penalty for MFA

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by minimizing the smoothness penalty of factor activation profiles, subject to constraints on VAF or BIC.

Usage

```
select_r_by_smoothness_mfa(
  list_of_data,
  K,
  rvec = 1:6,
  lambda_smooth = 0.001,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterFA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each ($T_i \times M$).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate.
<code>lambda_smooth</code>	Smoothness penalty weight. Default is $1e-3$.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>max_iter</code>	Maximum EM iterations.
<code>nIterFA</code>	Sub-iterations for FA update.
<code>tol</code>	Convergence tolerance.
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function fits MFA models for each r in `rvec` with fixed K , computes factor scores, and evaluates the smoothness penalty. The optimal r is selected by minimizing:

$$\text{score} = \text{BIC} + \lambda_{\text{smooth}} \times \text{smoothness}$$

among models satisfying $\text{VAF} \geq \text{vaf_threshold}$.

Value

A list with:

<code>best_r</code>	The selected r value.
<code>summary</code>	A data.frame with columns <code>r</code> , <code>BIC</code> , <code>VAF</code> , <code>smoothness</code> , <code>penalized_score</code> .
<code>best_model</code>	The fitted model for the selected r .

Examples

```
## Not run:
result <- select_r_by_smoothness_mfa(
  list_of_data, K = 3, rvec = 1:5,
  lambda_smooth = 1e-3, vaf_threshold = 0.9
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

```
select_r_by_smoothness_mPCA
```

Select Optimal r by Smoothness Penalty for MPCA

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by minimizing the smoothness penalty of PC activation profiles, subject to constraints on VAF or BIC.

Usage

```
select_r_by_smoothness_mPCA(
  list_of_data,
  K,
  rvec = 1:6,
  lambda_smooth = 0.001,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterPCA = 5,
```

```

    tol = 0.001,
    method = "EM",
    n_init = 1,
    use_kmeans_init = TRUE,
    subject_rdim_for_kmeans = 2,
    mc_cores = 1,
    n_threads = 1
  )

```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate.
<code>lambda_smooth</code>	Smoothness penalty weight. Default is 1e-3.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>max_iter</code>	Maximum EM iterations.
<code>nIterPCA</code>	Sub-iterations for PCA update.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form".
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function fits MPCA models for each r in `rvec` with fixed K, computes PC scores, and evaluates the smoothness penalty. The optimal r is selected by minimizing:

$$\text{score} = \text{BIC} + \lambda_{\text{smooth}} \times \text{smoothness}$$

among models satisfying $\text{VAF} \geq \text{vaf_threshold}$.

Value

A list with:

<code>best_r</code>	The selected r value.
<code>summary</code>	A data.frame with columns r, BIC, VAF, smoothness, penalized_score.
<code>best_model</code>	The fitted model for the selected r.

Examples

```
## Not run:
result <- select_r_by_smoothness_mpca(
  list_of_data, K = 3, rvec = 1:5,
  lambda_smooth = 1e-3, vaf_threshold = 0.9
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

select_r_by_vaf_mfa	<i>Select Minimum r Based on VAF Threshold for MFA</i>
---------------------	--

Description

This function implements a traditional EMG synergy approach: select the minimum r (number of factors) that achieves a target VAF (Variance Accounted For) threshold. This is commonly used in motor control research where VAF ≥ 90 -95 percent is desired.

Usage

```
select_r_by_vaf_mfa(
  list_of_data,
  K,
  rvec = 1:5,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterFA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>K</code>	Number of clusters (must be pre-specified).
<code>rvec</code>	Vector of candidate r values to test.
<code>vaf_threshold</code>	Target VAF threshold (default: 0.90 for 90 percent).
<code>max_iter</code>	Maximum EM iterations.
<code>nIterFA</code>	Sub-iterations within the factor-analyzer update.

tol	Convergence tolerance.
n_init	Number of random initializations.
use_kmeans_init	Whether to use k-means initialization.
subject_rdim_for_kmeans	Dimension for PCA-based features.
mc_cores	Number of CPU cores for parallelization.
n_threads	Number of OpenMP threads.

Details

This function fits MFA models for each r in `rvec` with fixed K , computes both VAF and BIC, and returns: 1. The minimum r that achieves the VAF threshold (traditional approach) 2. The optimal r by BIC (statistical approach)

Users can compare both criteria to make informed model selection decisions.

Value

A list containing: `summary` (data frame with r , VAF, BIC for each r), `best_r_vaf` (minimum r achieving VAF threshold), `best_r_bic` (optimal r by BIC), `best_model_vaf` (model with minimum r achieving threshold), and `best_model_bic` (model with optimal BIC).

`select_r_by_vaf_mPCA` *Select Minimum r Based on VAF Threshold for MPCA*

Description

This function implements a traditional EMG synergy approach: select the minimum r (number of principal components) that achieves a target VAF threshold. This is commonly used in motor control research where VAF ≥ 90 -95 percent is desired.

Usage

```
select_r_by_vaf_mPCA(
  list_of_data,
  K,
  rvec = 1:5,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of data matrices, each ($T_i \times M$).
<code>K</code>	Number of clusters (must be pre-specified).
<code>rvec</code>	Vector of candidate r values to test.
<code>vaf_threshold</code>	Target VAF threshold (default: 0.90 for 90 percent).
<code>max_iter</code>	Maximum EM iterations.
<code>nIterPCA</code>	Sub-iterations within the PCA/PPCA update.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form".
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for PCA-based features.
<code>mc_cores</code>	Number of CPU cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function fits MPCA models for each r in `rvec` with fixed K , computes both VAF and BIC, and returns: 1. The minimum r that achieves the VAF threshold (traditional approach) 2. The optimal r by BIC (statistical approach)

Users can compare both criteria to make informed model selection decisions.

Value

A list containing: `summary` (data frame with r , VAF, BIC for each r), `best_r_vaf` (minimum r achieving VAF threshold), `best_r_bic` (optimal r by BIC), `best_model_vaf` (model with minimum r achieving threshold), and `best_model_bic` (model with optimal BIC).

<code>select_r_by_vss_mfa</code>	<i>Select Optimal r by VSS Criterion for MFA</i>
----------------------------------	---

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by maximizing the VSS (Very Simple Structure) criterion, subject to constraints on VAF or BIC.

Usage

```
select_r_by_vss_mfa(
  list_of_data,
  K,
  rvec = 1:6,
  vss_threshold = 0.5,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterFA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate.
<code>vss_threshold</code>	Threshold for determining strong vs weak loadings. Default is 0.5.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>max_iter</code>	Maximum EM iterations.
<code>nIterFA</code>	Sub-iterations for FA update.
<code>tol</code>	Convergence tolerance.
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function fits MFA models for each r in rvec with fixed K, and evaluates the VSS criterion. The optimal r is selected by maximizing VSS among models satisfying VAF $\geq$ vaf_threshold.

Higher VSS values indicate better simple structure, where each muscle loads strongly on exactly one synergy.

Value

A list with:

best_r	The selected r value.
summary	A data.frame with columns r, BIC, VAF, VSS.
best_model	The fitted model for the selected r.

Examples

```
## Not run:
result <- select_r_by_vss_mfa(
  list_of_data, K = 3, rvec = 1:5,
  vss_threshold = 0.5, vaf_threshold = 0.9
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

select_r_by_vss_mPCA	<i>Select Optimal r by VSS Criterion for MPCA</i>
----------------------	---

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by maximizing the VSS (Very Simple Structure) criterion, subject to constraints on VAF or BIC.

Usage

```
select_r_by_vss_mPCA(
  list_of_data,
  K,
  rvec = 1:6,
  vss_threshold = 0.5,
  vaf_threshold = 0.9,
  max_iter = 50,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate.
<code>vss_threshold</code>	Threshold for determining strong vs weak loadings. Default is 0.5.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>max_iter</code>	Maximum EM iterations.
<code>nIterPCA</code>	Sub-iterations for PCA update.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form".
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function fits MPCA models for each r in `rvec` with fixed K, and evaluates the VSS criterion. The optimal r is selected by maximizing VSS among models satisfying `VAF >= vaf_threshold`. Higher VSS values indicate better simple structure, where each muscle loads strongly on exactly one synergy.

Value

A list with:

<code>best_r</code>	The selected r value.
<code>summary</code>	A data.frame with columns r, BIC, VAF, VSS.
<code>best_model</code>	The fitted model for the selected r.

Examples

```
## Not run:
result <- select_r_by_vss_mPCA(
  list_of_data, K = 3, rvec = 1:5,
  vss_threshold = 0.5, vaf_threshold = 0.9
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

select_r_clusterwise_mfa

Select Optimal r Using Cluster-wise Evaluation for MFA

Description

Selects the optimal number of factors (r) by evaluating each cluster separately, rather than averaging metrics across all clusters. This approach is useful when different clusters may have different optimal r values.

Usage

```
select_r_clusterwise_mfa(
  list_of_data,
  K,
  rvec = 1:6,
  criteria = c("VAF", "VSS", "smoothness"),
  agg_method = c("max", "mean", "min", "majority"),
  vaf_threshold = NULL,
  vss_threshold = 0.5,
  vss_method = c("threshold", "ratio"),
  vss_rotate = c("none", "varimax"),
  lambda_smooth = 0.001,
  weights = c(VAF = 0.4, VSS = 0.4, smoothness = 0.2),
  vss_normalize = c("logr", "max", "r", "r2", "none"),
  w_elbow = 0,
  w_r = 0,
  vaf_degeneracy_threshold = 0.6,
  max_iter = 50,
  nIterFA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1,
  verbose = FALSE,
  mode = c("clusterwise", "global-first"),
  global_model = c("PPCA", "FA"),
  global_criterion = c("CV", "ParallelAnalysis"),
  global_folds = 5,
  global_seed = NULL,
  r_init = NULL
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.
<code>criteria</code>	Character vector specifying which criteria to use for selection. Options: "VAF", "VSS", "smoothness". Default is c("VAF", "VSS", "smoothness").
<code>agg_method</code>	Method for aggregating cluster-specific r values. Options: "max" (conservative, takes maximum r across clusters), "mean" (average r, rounded), "min" (minimum r across clusters), "majority" (most common r value). Default is "max".
<code>vaf_threshold</code>	Minimum VAF threshold for each cluster. Set to NULL to disable hard filtering. Default is NULL (disabled).
<code>vss_threshold</code>	Threshold for VSS criterion. Default is 0.5.
<code>vss_method</code>	Method for computing VSS. Options: "threshold" (count variables with exactly one strong loading above threshold), "ratio" (compute concentration ratio $\max(\lambda^2)/\sum(\lambda^2)$ for each variable). Default is "threshold". The "ratio" method is more robust when loadings are small in absolute magnitude.
<code>vss_rotate</code>	Rotation method to apply before computing VSS. Options: "none" (no rotation), "varimax" (orthogonal rotation). Default is "none". VSS on unrotated FA loadings is rotation-dependent and often uninformative. Rotation is recommended for meaningful VSS.
<code>lambda_smooth</code>	Smoothness penalty weight (deprecated, use weights instead). Default is 1e-3.
<code>weights</code>	Named vector of weights for combining criteria. Default is c(VAF=0.4, VSS=0.4, smoothness=0.2).
<code>vss_normalize</code>	Method for normalizing VSS to account for r-dependence. Options: "logr" (divide by log(r)), "max" (divide by max VSS), "r" (divide by r), "r2" (divide by r ²), "none". Default is "logr".
<code>w_elbow</code>	Weight for DeltaVAF (elbow detection). Higher values emphasize diminishing returns in VAF. Default is 0 (disabled).
<code>w_r</code>	Weight for direct r penalty (AIC/BIC style). Higher values penalize larger r. Default is 0 (disabled).
<code>vaf_degeneracy_threshold</code>	Threshold for detecting degenerate clusters. If max VAF across all r values is below this threshold, the cluster is considered degenerate and a warning is issued. Default is 0.6.
<code>max_iter</code>	Maximum EM iterations for initial clustering.
<code>nIterFA</code>	Sub-iterations for FA update.
<code>tol</code>	Convergence tolerance.
<code>n_init</code>	Number of random initializations for initial clustering.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.

mc_cores	Number of cores for parallelization.
n_threads	Number of OpenMP threads.
verbose	Logical. If TRUE, print detailed diagnostic information during r selection, including per-r metrics for each cluster and warnings about NA values or degenerate clusters. Default is FALSE.
mode	Selection mode. Options: "clusterwise" (default, evaluate r separately per cluster), "global-first" (first estimate global r using FA/PPCA on pooled data, then use for MFA). The "global-first" mode is recommended as it provides a cluster-independent baseline.
global_model	Model for global r estimation (only used when mode = "global-first"). Options: "PPCA" (default), "FA". PPCA is more stable for mixture models.
global_criterion	Criterion for global r selection (only used when mode = "global-first"). Options: "CV" (default), "ParallelAnalysis". CV is the primary criterion.
global_folds	Number of CV folds for global r selection (only used when mode = "global-first"). Default is 5.
global_seed	Random seed for global r selection (only used when mode = "global-first"). Default is NULL.
r_init	Initial r value for the first-stage clustering (only used when mode = "clusterwise"). If NULL (default), uses median(rvec). When called from hybrid mode in select_r_mixture(), this is set to r_global to use the global estimate as initialization.

Details

This function implements a two-stage approach:

1. Fit an initial MFA model with K clusters and a reasonable r value (using the median of rvec) to obtain cluster assignments.
2. For each cluster k, subset the data to subjects assigned to that cluster, and evaluate different r values using cluster-specific metrics.
3. Select the optimal r_k for each cluster based on normalized scoring that combines VAF (higher is better), VSS (higher is better, normalized by log(r)), and smoothness (lower is better) with specified weights.
4. Aggregate the cluster-specific r values using the specified method.

The scoring approach addresses the issue where VAF monotonically increases with r by normalizing all metrics per cluster and using weighted combination. VSS is normalized by log(r) to account for the fact that more factors naturally lead to more complex structures. Smoothness acts as a penalty (subtracted from score).

The cluster-wise approach is particularly useful when:

- Different motor control strategies require different numbers of synergies
- Averaging metrics across clusters masks cluster-specific structure
- You want to understand the heterogeneity in synergy complexity

Important Note: The cluster-wise MFA(K=1) approach is experimental and can be unstable. Each cluster is evaluated independently using a single-cluster MFA model, which may not capture the full complexity of the data. Use with caution and carefully inspect the verbose output and degeneracy warnings.

Value

A list with:

cluster_r	Named vector of optimal r for each cluster.
agg_r	Aggregated r value across clusters.
cluster_metrics	List of data frames with metrics for each cluster.
initial_fit	The initial MFA fit used to determine cluster assignments.
agg_method	The aggregation method used.
mode	The selection mode used.
global_result	(Only for mode = "global-first") The result from select_r_global_mfa().

Examples

```
## Not run:
# Basic usage with default weights (clusterwise mode)
result <- select_r_clusterwise_mfa(
  list_of_data, K = 3, rvec = 2:6,
  criteria = c("VAF", "VSS", "smoothness")
)
print(result$cluster_r) # r for each cluster
print(result$agg_r)     # aggregated r

# Global-first mode (recommended)
result <- select_r_clusterwise_mfa(
  list_of_data, K = 3, rvec = 2:6,
  mode = "global-first",
  global_model = "PPCA",
  global_criterion = "CV"
)
print(result$agg_r) # global r estimate

# Custom weights emphasizing VAF (clusterwise mode)
result <- select_r_clusterwise_mfa(
  list_of_data, K = 3, rvec = 2:6,
  mode = "clusterwise",
  criteria = c("VAF", "VSS", "smoothness"),
  weights = c(VAF = 0.6, VSS = 0.3, smoothness = 0.1)
)

## End(Not run)
```

select_r_clusterwise_mPCA

Select Optimal r Using Cluster-wise Evaluation for MPCA

Description

Selects the optimal number of principal components (r) by evaluating each cluster separately, rather than averaging metrics across all clusters.

Usage

```
select_r_clusterwise_mPCA(
  list_of_data,
  K,
  rvec = 1:6,
  criteria = c("VAF", "VSS", "smoothness"),
  agg_method = c("max", "mean", "min", "majority"),
  vaf_threshold = NULL,
  vss_threshold = 0.5,
  vss_method = c("threshold", "ratio"),
  vss_rotate = c("none", "varimax"),
  lambda_smooth = 0.001,
  weights = c(VAF = 0.4, VSS = 0.4, smoothness = 0.2),
  vss_normalize = c("logr", "max", "r", "r2", "none"),
  w_elbow = 0,
  w_r = 0,
  vaf_degeneracy_threshold = 0.6,
  max_iter = 50,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1,
  verbose = FALSE,
  mode = c("clusterwise", "global-first"),
  global_model = c("PPCA", "FA"),
  global_criterion = c("CV", "ParallelAnalysis"),
  global_folds = 5,
  global_seed = NULL,
  r_init = NULL
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each ($T_i \times M$).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.
<code>criteria</code>	Character vector specifying which criteria to use for selection. Options: "VAF", "VSS", "smoothness". Default is c("VAF", "VSS", "smoothness").
<code>agg_method</code>	Method for aggregating cluster-specific r values. Options: "max", "mean", "min", "majority". Default is "max".
<code>vaf_threshold</code>	Minimum VAF threshold for each cluster. Set to NULL to disable hard filtering. Default is NULL (disabled).
<code>vss_threshold</code>	Threshold for VSS criterion. Default is 0.5.
<code>vss_method</code>	Method for computing VSS. Options: "threshold" (count variables with exactly one strong loading above threshold), "ratio" (compute concentration ratio $\max(\lambda^2)/\sum(\lambda^2)$ for each variable). Default is "threshold". The "ratio" method is more robust when loadings are small in absolute magnitude.
<code>vss_rotate</code>	Rotation method to apply before computing VSS. Options: "none" (no rotation), "varimax" (orthogonal rotation). Default is "none". VSS on unrotated FA loadings is rotation-dependent and often uninformative. Rotation is recommended for meaningful VSS.
<code>lambda_smooth</code>	Smoothness penalty weight (deprecated, use weights instead). Default is 1e-3.
<code>weights</code>	Named vector of weights for combining criteria. Default is c(VAF=0.4, VSS=0.4, smoothness=0.2).
<code>vss_normalize</code>	Method for normalizing VSS to account for r-dependence. Options: "logr" (divide by $\log(r)$), "max" (divide by max VSS), "r" (divide by r), "r2" (divide by r^2), "none". Default is "logr".
<code>w_elbow</code>	Weight for DeltaVAF (elbow detection). Higher values emphasize diminishing returns in VAF. Default is 0 (disabled).
<code>w_r</code>	Weight for direct r penalty (AIC/BIC style). Higher values penalize larger r. Default is 0 (disabled).
<code>vaf_degeneracy_threshold</code>	Threshold for detecting degenerate clusters. If max VAF across all r values is below this threshold, the cluster is considered degenerate and a warning is issued. Default is 0.6.
<code>max_iter</code>	Maximum EM iterations for initial clustering.
<code>nIterPCA</code>	Sub-iterations for PCA update.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form".
<code>n_init</code>	Number of random initializations for initial clustering.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.

mc_cores	Number of cores for parallelization.
n_threads	Number of OpenMP threads.
verbose	Logical. If TRUE, print detailed diagnostic information during r selection, including per-r metrics for each cluster and warnings about NA values or degenerate clusters. Default is FALSE.
mode	Selection mode. Options: "clusterwise" (default, evaluate r separately per cluster), "global-first" (first estimate global r using FA/PPCA on pooled data, then use for MPCA). The "global-first" mode is recommended as it provides a cluster-independent baseline.
global_model	Model for global r estimation (only used when mode = "global-first"). Options: "PPCA" (default), "FA". PPCA is more stable for mixture models.
global_criterion	Criterion for global r selection (only used when mode = "global-first"). Options: "CV" (default), "ParallelAnalysis". CV is the primary criterion.
global_folds	Number of CV folds for global r selection (only used when mode = "global-first"). Default is 5.
global_seed	Random seed for global r selection (only used when mode = "global-first"). Default is NULL.
r_init	Initial r value for the first-stage clustering (only used when mode = "clusterwise"). If NULL (default), uses median(rvec). When called from hybrid mode in select_r_mixture(), this is set to r_global to use the global estimate as initialization.

Details

This function implements the same two-stage approach as [select_r_clusterwise_mfa](#), but for Mixture PCA models. See [select_r_clusterwise_mfa](#) for details on the scoring approach.

Important Note: The cluster-wise MPCA(K=1) approach is experimental and can be unstable. Each cluster is evaluated independently using a single-cluster MPCA model, which may not capture the full complexity of the data. Use with caution and carefully inspect the verbose output and degeneracy warnings.

Value

A list with:

cluster_r	Named vector of optimal r for each cluster.
agg_r	Aggregated r value across clusters.
cluster_metrics	List of data frames with metrics for each cluster.
initial_fit	The initial MPCA fit used to determine cluster assignments.
agg_method	The aggregation method used.
mode	The selection mode used.
global_result	(Only for mode = "global-first") The result from select_r_global_mfa().

Examples

```
## Not run:
result <- select_r_clusterwise_mpca(
  list_of_data, K = 3, rvec = 2:6,
  criteria = c("VAF", "VSS", "smoothness")
)
print(result$cluster_r)
print(result$agg_r)

## End(Not run)
```

select_r_global_mfa	<i>Select Optimal r Using Global Model (FA or PPCA) with Cross-Validation</i>
---------------------	---

Description

Estimates the optimal number of factors (r) using a global Factor Analysis (FA) or Probabilistic PCA (PPCA) model on pooled data. This function is designed for use before fitting mixture models (MFA/MPCA), as it provides a cluster-independent estimate of r based on the overall data structure.

Usage

```
select_r_global_mfa(
  list_of_data,
  rvec = 1:6,
  model = c("PPCA", "FA"),
  criterion = c("CV", "ParallelAnalysis"),
  folds = 5,
  cv_metric = c("negLogLik", "frobenius"),
  n_permutations = 100,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

list_of_data	A list of N matrices, each ($T_i \times M$).
rvec	Vector of candidate r values to evaluate. Default is 1:6.
model	Model type for global r estimation. Options: "PPCA" (Probabilistic PCA, default), "FA" (Factor Analysis). PPCA is recommended as it is more stable and theory-consistent for mixture models.
criterion	Selection criterion. Options: "CV" (Cross-Validation, default), "ParallelAnalysis" (Parallel Analysis). CV is the primary criterion as it is stable and works for both FA and PPCA. Parallel Analysis is optional and lower priority.

<code>folds</code>	Number of folds for cross-validation. Default is 5. CV is performed at the subject level (not time-window level) to respect the generative structure.
<code>cv_metric</code>	Metric for CV evaluation. Options: "negLogLik" (negative log-likelihood, default), "frobenius" (Frobenius norm of reconstruction error). Negative log-likelihood is the principled choice for probabilistic models.
<code>n_permutations</code>	Number of permutations for Parallel Analysis. Default is 100. Only used when <code>criterion = "ParallelAnalysis"</code> .
<code>seed</code>	Random seed for reproducibility (CV fold assignment and PA permutations). Default is NULL (no seed set).
<code>verbose</code>	Logical. If TRUE, print detailed diagnostic information during r selection. Default is FALSE.

Details

This function implements model-based r estimation for use before mixture model fitting. The key design principles are:

Why Global Model? The global r must be estimated using FA or PPCA on pooled data, NOT using MFA($K > 1$). Estimating r from a mixture model is cluster-dependent and unstable, as different cluster configurations yield different r estimates. A global model provides a cluster-independent baseline.

Cross-Validation Design: CV is performed at the subject level, not time-window level. Time-window CV would leak subject-level covariance structure and provide overly optimistic estimates. Subject-level CV properly reflects the generative structure where subjects are the independent units.

For each fold:

1. Split subjects into training and test sets
2. Fit FA/PPCA on training data (pooled across subjects)
3. Evaluate reconstruction error or log-likelihood on test data
4. Average across folds to get CV error for each r

The optimal r is selected by minimizing CV error (or using the one-standard-error rule for parsimony).

Parallel Analysis: PA compares observed eigenvalues to those from permuted data. The optimal r is where observed eigenvalues exceed the 95th percentile of permuted eigenvalues. Note: PA is theoretically questionable for FA (as it assumes isotropic noise), so CV is preferred.

Value

A list with:

<code>r_global</code>	The selected optimal r value.
<code>summary</code>	A data.frame with columns <code>r</code> and criterion-specific metrics (e.g., <code>cv_error</code> , <code>cv_se</code>).
<code>criterion</code>	The criterion used for selection.
<code>model</code>	The model type used (FA or PPCA).
<code>cv_curve</code>	For CV: ggplot object showing CV error vs r (if ggplot2 available).

vaf_curve	For CV: ggplot object showing VAF vs r (if ggplot2 available).
pa_curve	For PA: ggplot object showing observed vs permuted eigenvalues (if ggplot2 available).
diagnostics	Additional diagnostic information.

Examples

```
## Not run:
# Basic usage with default PPCA and CV
result <- select_r_global_mfa(
  list_of_data, rvec = 2:6,
  model = "PPCA", criterion = "CV", folds = 5
)
print(result$r_global)
print(result$summary)

# With FA model
result_fa <- select_r_global_mfa(
  list_of_data, rvec = 2:6,
  model = "FA", criterion = "CV"
)

# Using Parallel Analysis
result_pa <- select_r_global_mfa(
  list_of_data, rvec = 2:6,
  criterion = "ParallelAnalysis", n_permutations = 100
)

## End(Not run)
```

select_r_global_mPCA	<i>Select Optimal r Using Global Model (PPCA) with Cross-Validation for MPCA</i>
----------------------	--

Description

Estimates the optimal number of principal components (r) using a global Probabilistic PCA (PPCA) model on pooled data. This function is designed for use before fitting mixture MPCA models, as it provides a cluster-independent estimate of r based on the overall data structure.

Usage

```
select_r_global_mPCA(
  list_of_data,
  rvec = 1:6,
  model = c("PPCA", "FA"),
  criterion = c("CV", "ParallelAnalysis"),
```

```

    folds = 5,
    cv_metric = c("negLogLik", "frobenius"),
    n_permutations = 100,
    seed = NULL,
    verbose = FALSE
  )

```

Arguments

<code>list_of_data</code>	A list of N matrices, each ($T_i \times M$).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.
<code>model</code>	Model type for global r estimation. Options: "PPCA" (Probabilistic PCA, default), "FA" (Factor Analysis). PPCA is recommended as it is more stable and theory-consistent for mixture models.
<code>criterion</code>	Selection criterion. Options: "CV" (Cross-Validation, default), "ParallelAnalysis" (Parallel Analysis). CV is the primary criterion as it is stable and works for both FA and PPCA. Parallel Analysis is optional and lower priority.
<code>folds</code>	Number of folds for cross-validation. Default is 5. CV is performed at the subject level (not time-window level) to respect the generative structure.
<code>cv_metric</code>	Metric for CV evaluation. Options: "negLogLik" (negative log-likelihood, default), "frobenius" (Frobenius norm of reconstruction error). Negative log-likelihood is the principled choice for probabilistic models.
<code>n_permutations</code>	Number of permutations for Parallel Analysis. Default is 100. Only used when <code>criterion = "ParallelAnalysis"</code> .
<code>seed</code>	Random seed for reproducibility (CV fold assignment and PA permutations). Default is NULL (no seed set).
<code>verbose</code>	Logical. If TRUE, print detailed diagnostic information during r selection. Default is FALSE.

Details

This function implements model-based r estimation for use before mixture MPCA fitting. The key design principles are:

Why Global Model? The global r must be estimated using PPCA (or FA) on pooled data, NOT using MPCA($K > 1$). Estimating r from a mixture model is cluster-dependent and unstable, as different cluster configurations yield different r estimates. A global model provides a cluster-independent baseline.

Cross-Validation Design: CV is performed at the subject level, not time-window level. Time-window CV would leak subject-level covariance structure and provide overly optimistic estimates. Subject-level CV properly reflects the generative structure where subjects are the independent units.

For each fold:

1. Split subjects into training and test sets
2. Fit PPCA/FA on training data (pooled across subjects)
3. Evaluate reconstruction error or log-likelihood on test data

4. Average across folds to get CV error for each r

The optimal r is selected by minimizing CV error (or using the one-standard-error rule for parsimony).

Parallel Analysis: PA compares observed eigenvalues to those from permuted data. The optimal r is where observed eigenvalues exceed the 95th percentile of permuted eigenvalues. Note: PA is theoretically questionable for FA (as it assumes isotropic noise), so CV is preferred.

Value

A list with:

r_global	The selected optimal r value.
summary	A data.frame with columns r and criterion-specific metrics (e.g., cv_error, cv_se).
criterion	The criterion used for selection.
model	The model type used (FA or PPCA).
cv_curve	For CV: ggplot object showing CV error vs r (if ggplot2 available).
vaf_curve	For CV: ggplot object showing VAF vs r (if ggplot2 available).
pa_curve	For PA: ggplot object showing observed vs permuted eigenvalues (if ggplot2 available).
diagnostics	Additional diagnostic information.

Examples

```
## Not run:
# Basic usage with default PPCA and CV
result <- select_r_global_mpca(
  list_of_data, rvec = 2:6,
  model = "PPCA", criterion = "CV", folds = 5
)
print(result$r_global)
print(result$summary)

# With FA model
result_fa <- select_r_global_mpca(
  list_of_data, rvec = 2:6,
  model = "FA", criterion = "CV"
)

# Using Parallel Analysis
result_pa <- select_r_global_mpca(
  list_of_data, rvec = 2:6,
  criterion = "ParallelAnalysis", n_permutations = 100
)

## End(Not run)
```

select_r_mixture

*Unified r Selection for Mixture Models (MFA/MPCA)***Description**

This function provides a unified interface for selecting the optimal number of factors (r) in mixture models. It supports three modes: "clusterwise" (evaluate r separately per cluster), "global-first" (estimate r from pooled data first), and "hybrid" (combine both approaches).

Usage

```
select_r_mixture(
  list_of_data,
  K,
  rvec = 1:6,
  mode = c("clusterwise", "global-first", "hybrid"),
  model_type = c("MFA", "MPCA"),
  criteria = c("VAF", "VSS", "smoothness"),
  agg_method = c("max", "mean", "min", "majority", "global"),
  global_model = c("PPCA", "FA"),
  global_criterion = c("CV", "ParallelAnalysis"),
  global_folds = 5,
  global_seed = NULL,
  weights = c(VAF = 0.4, VSS = 0.4, smoothness = 0.2),
  vss_threshold = 0.5,
  vss_method = c("threshold", "ratio"),
  vss_rotate = c("none", "varimax"),
  vss_normalize = c("logr", "max", "r", "r2", "none"),
  max_iter = 50,
  nIterFA = 5,
  nIterPCA = 5,
  tol = 0.001,
  n_init = 1,
  use_kmeans_init = TRUE,
  subject_rdim_for_kmeans = 2,
  mc_cores = 1,
  n_threads = 1,
  verbose = FALSE,
  refit_final = TRUE
)
```

Arguments

<code>list_of_data</code>	A list of N matrices, each ($T_i \times M$).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.

mode	Selection mode. Options: • "clusterwise" (default): Evaluate r separately for each cluster, then aggregate • "global-first": Estimate r using global FA/PPCA on pooled data • "hybrid": Use global r as initialization for clusterwise evaluation
model_type	Model type. Options: "MFA" (Mixture Factor Analysis, default), "MPCA" (Mixture Principal Component Analysis).
criteria	Character vector specifying which criteria to use for clusterwise selection. Options: "VAF", "VSS", "smoothness". Default is c("VAF", "VSS", "smoothness"). Only used when mode = "clusterwise" or "hybrid".
agg_method	Method for aggregating cluster-specific r values. Options: "max" (conservative, takes maximum r across clusters), "mean" (average r, rounded), "min" (minimum r across clusters), "majority" (most common r value), "global" (use the global r estimate r_global as the final r; only valid in hybrid mode). Default is "max". In clusterwise mode, only "max", "mean", "min", "majority" are allowed. In hybrid mode, you may additionally use "global" to set r_final = r_global. Ignored when mode = "global-first".
global_model	Model for global r estimation. Options: "PPCA" (default), "FA". Only used when mode = "global-first" or "hybrid".
global_criterion	Criterion for global r selection. Options: "CV" (default), "ParallelAnalysis". Only used when mode = "global-first" or "hybrid".
global_folds	Number of CV folds for global r selection. Default is 5. Only used when mode = "global-first" or "hybrid" with criterion = "CV".
global_seed	Random seed for global r selection. Default is NULL.
weights	Named vector of weights for combining criteria in clusterwise mode. Default is c(VAF=0.4, VSS=0.4, smoothness=0.2).
vss_threshold	Threshold for VSS criterion. Default is 0.5.
vss_method	Method for computing VSS. Options: "threshold", "ratio". Default is "threshold".
vss_rotate	Rotation method for VSS. Options: "none", "varimax". Default is "none".
vss_normalize	Method for normalizing VSS. Options: "logr", "max", "r", "r2", "none". Default is "logr".
max_iter	Maximum EM iterations for model fitting.
nIterFA	Sub-iterations for FA update (MFA only).
nIterPCA	Sub-iterations for PCA update (MPCA only).
tol	Convergence tolerance.
n_init	Number of random initializations for mixture model fitting.
use_kmeans_init	Whether to use k-means initialization.
subject_rdim_for_kmeans	Dimension for k-means features.

mc_cores	Number of cores for parallelization.
n_threads	Number of OpenMP threads.
verbose	Logical. If TRUE, print detailed diagnostic information.
refit_final	Logical. If TRUE, refit the final model with selected r_final. Default is TRUE.

Details

This function provides a unified interface for r selection in mixture models, integrating three complementary approaches:

Mode: "global-first"

1. Pool all data and fit a global FA/PPCA model
2. Select r using cross-validation or parallel analysis
3. Return r_global as the final estimate

This mode is fast and provides a cluster-independent baseline. It's recommended when you want a simple, stable estimate that doesn't depend on cluster structure.

Mode: "clusterwise"

1. Fit initial mixture model with median(rvec) to get cluster assignments
2. For each cluster, evaluate different r values using cluster-specific metrics
3. Aggregate cluster-specific r values using specified method

This mode is useful when different clusters may require different numbers of factors.

Mode: "hybrid"

1. Estimate r_global using global FA/PPCA (as in "global-first")
2. Use r_global as initialization for clusterwise evaluation
3. Evaluate r values around r_global for each cluster
4. Aggregate cluster-specific r values

This mode combines the stability of global estimation with the flexibility of cluster-specific evaluation.

Value

A list with class "mixture_r_selection" containing:

r_final	The final selected r value.
r_global	Global r estimate (if mode = "global-first" or "hybrid").
r_cluster	Named vector of cluster-specific r values (if mode = "clusterwise" or "hybrid").
initial_fit	Initial mixture model fit used for cluster assignments (if applicable).
final_fit	Final mixture model fit with r_final (if refit_final = TRUE).
diagnostics	List containing detailed diagnostic information.
mode	The selection mode used.
model_type	The model type used (MFA or MPCA).
agg_method	The aggregation method used (if applicable).

Examples

```
## Not run:
# Global-first mode (recommended for simplicity)
result <- select_r_mixture(
  list_of_data, K = 3, rvec = 2:6,
  mode = "global-first",
  model_type = "MFA"
)
print(result$r_final)

# Clusterwise mode
result <- select_r_mixture(
  list_of_data, K = 3, rvec = 2:6,
  mode = "clusterwise",
  model_type = "MFA",
  agg_method = "max"
)

# Hybrid mode
result <- select_r_mixture(
  list_of_data, K = 3, rvec = 2:6,
  mode = "hybrid",
  model_type = "MPCA"
)

## End(Not run)
```

select_r_multicriteria_mfa

Select Optimal r Using Multi-Criteria Approach for MFA

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by combining multiple criteria: BIC, VAF, smoothness penalty, and VSS.

Usage

```
select_r_multicriteria_mfa(
  list_of_data,
  K,
  rvec = 1:6,
  vaf_threshold = 0.9,
  lambda_smooth = 0.001,
  vss_threshold = 0.5,
  weights = c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1),
  max_iter = 50,
```

```

nIterFA = 5,
tol = 0.001,
n_init = 1,
use_kmeans_init = TRUE,
subject_rdim_for_kmeans = 2,
mc_cores = 1,
n_threads = 1
)

```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>lambda_smooth</code>	Smoothness penalty weight. Default is 1e-3.
<code>vss_threshold</code>	Threshold for VSS criterion. Default is 0.5.
<code>weights</code>	Named vector of weights for each criterion. Default is c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1).
<code>max_iter</code>	Maximum EM iterations.
<code>nIterFA</code>	Sub-iterations for FA update.
<code>tol</code>	Convergence tolerance.
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function combines multiple criteria to select r:

- BIC: Lower is better (penalizes complexity)
- VAF: Higher is better (rewards fit quality)
- Smoothness: Lower is better (penalizes wiggly factors)
- VSS: Higher is better (rewards simple structure)

Each criterion is normalized to [0, 1] range, then combined using weights:

$$\text{score} = w_{\text{BIC}} \times \text{BIC}_{\text{norm}} + w_{\text{VAF}} \times (1 - \text{VAF}_{\text{norm}}) + w_{\text{smooth}} \times \text{smooth}_{\text{norm}} + w_{\text{VSS}} \times (1 - \text{VSS}_{\text{norm}})$$

The r with the minimum composite score is selected among models satisfying VAF >= vaf_threshold.

Value

A list with:

best_r	The selected r value.
summary	A data.frame with columns r, BIC, VAF, smoothness, VSS, composite_score.
best_model	The fitted model for the selected r.

Examples

```
## Not run:
result <- select_r_multicriteria_mfa(
  list_of_data, K = 3, rvec = 1:5,
  vaf_threshold = 0.9,
  lambda_smooth = 1e-3,
  weights = c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1)
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

```
select_r_multicriteria_mPCA
```

Select Optimal r Using Multi-Criteria Approach for MPCA

Description

Selects the optimal number of factors (r) for a given number of clusters (K) by combining multiple criteria: BIC, VAF, smoothness penalty, and VSS.

Usage

```
select_r_multicriteria_mPCA(
  list_of_data,
  K,
  rvec = 1:6,
  vaf_threshold = 0.9,
  lambda_smooth = 0.001,
  vss_threshold = 0.5,
  weights = c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1),
  max_iter = 50,
  nIterPCA = 5,
  tol = 0.001,
  method = "EM",
  n_init = 1,
  use_kmeans_init = TRUE,
```

```

    subject_rdim_for_kmeans = 2,
    mc_cores = 1,
    n_threads = 1
)

```

Arguments

<code>list_of_data</code>	A list of N matrices, each (T _i x M).
<code>K</code>	Number of clusters (fixed).
<code>rvec</code>	Vector of candidate r values to evaluate. Default is 1:6.
<code>vaf_threshold</code>	Minimum VAF threshold. Models with VAF below this are excluded. Default is 0.9.
<code>lambda_smooth</code>	Smoothness penalty weight. Default is 1e-3.
<code>vss_threshold</code>	Threshold for VSS criterion. Default is 0.5.
<code>weights</code>	Named vector of weights for each criterion. Default is c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1).
<code>max_iter</code>	Maximum EM iterations.
<code>nIterPCA</code>	Sub-iterations for PCA update.
<code>tol</code>	Convergence tolerance.
<code>method</code>	"EM" or "closed_form".
<code>n_init</code>	Number of random initializations.
<code>use_kmeans_init</code>	Whether to use k-means initialization.
<code>subject_rdim_for_kmeans</code>	Dimension for k-means features.
<code>mc_cores</code>	Number of cores for parallelization.
<code>n_threads</code>	Number of OpenMP threads.

Details

This function combines multiple criteria to select r:

- BIC: Lower is better (penalizes complexity)
- VAF: Higher is better (rewards fit quality)
- Smoothness: Lower is better (penalizes wiggly factors)
- VSS: Higher is better (rewards simple structure)

Each criterion is normalized to [0, 1] range, then combined using weights:

$$\text{score} = w_{\text{BIC}} \times \text{BIC}_{\text{norm}} + w_{\text{VAF}} \times (1 - \text{VAF}_{\text{norm}}) + w_{\text{smooth}} \times \text{smooth}_{\text{norm}} + w_{\text{VSS}} \times (1 - \text{VSS}_{\text{norm}})$$

The r with the minimum composite score is selected among models satisfying $\text{VAF} \geq \text{vaf_threshold}$.

Value

A list with:

best_r	The selected r value.
summary	A data.frame with columns r, BIC, VAF, smoothness, VSS, composite_score.
best_model	The fitted model for the selected r.

Examples

```
## Not run:
result <- select_r_multicriteria_mpc(
  list_of_data, K = 3, rvec = 1:5,
  vaf_threshold = 0.9,
  lambda_smooth = 1e-3,
  weights = c(BIC = 1, VAF = 1, smoothness = 1, VSS = 1)
)
print(result$summary)
best_model <- result$best_model

## End(Not run)
```

setup_future_backend *Setup Future Backend for HPC*

Description

Configures the future backend for parallel execution. Supports local multicore, PBS/TORQUE, SLURM, and other HPC schedulers via future.batchtools.

Usage

```
setup_future_backend(
  backend = c("multisession", "multicore", "cluster", "sequential", "batchtools_torque",
    "batchtools_slurm", "batchtools_sge", "batchtools_lsf"),
  workers = NULL,
  resources = list(),
  template = NULL,
  verbose = TRUE
)
```

Arguments

backend Character. Backend type:

- "multisession": Local parallel sessions (default)
- "multicore": Local forked processes (Unix only)
- "cluster": Manual cluster setup

	• "batchtools_torque": PBS/TORQUE scheduler • "batchtools_slurm": SLURM scheduler • "batchtools_sge": SGE scheduler • "batchtools_lsf": LSF scheduler
workers	Integer. Number of workers for local backends.
resources	List. HPC resource specifications for batchtools backends: • walltime: Wall time limit (e.g., "04:00:00") • memory: Memory per job (e.g., "4gb" or "4096") • ncpus: CPUs per job • queue: Queue/partition name
template	Character. Path to batchtools template file (optional).
verbose	Logical. Print setup information.

Details

This function is a wrapper around `future::plan()` that simplifies HPC backend configuration. For PBS/TORQUE, it uses `future.batchtools::batchtools_torque`.

Value

Invisibly returns the previous plan.

Examples

```
## Not run:
# Local multicore
setup_future_backend("multisession", workers = 4)

# PBS/TORQUE with resources
setup_future_backend(
  "batchtools_torque",
  resources = list(walltime = "02:00:00", memory = "8gb")
)

# Reset to sequential
setup_future_backend("sequential")

## End(Not run)
```

show_parallel_options *Print Summary of Available Parallelization Options*

Description

Displays a summary of available parallelization strategies and their current availability status.

Usage

```
show_parallel_options()
```

Value

Invisibly returns a list with availability status.

Examples

```
show_parallel_options()
```

simulate_dynamic_synergy_data
Simulate Dynamic Synergy Data with Three Orthogonal Control Axes

Description

This function extends `simulate_dynamic_synergy_data()` to systematically evaluate how different motor control strategies affect muscle synergy structures. The simulation model includes three orthogonal control axes: (1) Spatial structure, (2) Temporal coordination, and (3) Stability-Adaptability. These axes represent complementary aspects of motor control diversity observed in both rehabilitation and sports science contexts.

Usage

```
simulate_dynamic_synergy_data(
  N = 30,
  K = 3,
  r = 3,
  M = 8,
  T_each = 200,
  cluster_sep_spatial = 1,
  cluster_sep_temporal = 1,
  cluster_sep_stability = 1,
  freq_range = c(0.5, 3),
  amp_range = c(0.5, 1.5),
  base_level = 0.05,
```

```

    rectify_output = TRUE,
    seed = 123
)

```

Arguments

N	Number of subjects (time series). Default is 30.
K	Number of clusters (subgroups). Default is 3.
r	Number of synergies per cluster. Default is 3.
M	Number of observed channels (muscles). Default is 8.
T_each	Time-series length per subject. Default is 200.
cluster_sep_spatial	Numeric scalar (0-2) controlling spatial structure differentiation between clusters. Higher values create more distinct spatial modules. Default is 1.0.
cluster_sep_temporal	Numeric scalar (0-2) controlling temporal coordination differentiation between clusters. Higher values create more distinct temporal patterns. Default is 1.0.
cluster_sep_stability	Numeric scalar (0-2) controlling stability-adaptability differentiation between clusters. Higher values create more distinct noise profiles. Default is 1.0.
freq_range	Numeric vector of length 2 specifying the range of base frequencies (in Hz) for the periodic activation signals. Default is c(0.5, 3.0).
amp_range	Numeric vector of length 2 specifying the range of amplitudes for the periodic activation signals. Default is c(0.5, 1.5).
base_level	Numeric scalar specifying the baseline activation level added to C(t) to avoid all-zero activations. Default is 0.05.
rectify_output	Logical; if TRUE, apply half-wave rectification to X(t) to produce non-negative EMG-like signals. Default is TRUE.
seed	Random seed for reproducibility. Default is 123.

Details

Each cluster ($k = 1, \dots, K$) is assigned a unique combination of parameters along these three axes, emulating distinct movement control strategies (e.g., proximal-dominant, alternating coordination, or adaptive noisy control).

Value

A list with the following components:

- X_list List of length N; each element is a ($T_each \times M$) matrix of generated EMG signals.
- C_list List of length N; each element is a ($T_each \times r$) matrix of synergy activation signals.
- Lambda_list List of length K; each element is a ($M \times r$) matrix of cluster-specific synergy loadings (non-negative).
- z_true Integer vector of length N indicating true cluster assignments (1 to K).

`spatial_params` List containing spatial structure parameters for each cluster (bias patterns).

`temporal_params` List containing temporal coordination parameters for each cluster (frequencies, phase offsets).

`stability_params` List containing stability-adaptability parameters for each cluster (noise levels, filter lengths).

Spatial Structure Axis

The spatial axis defines how strongly each muscle group contributes to each synergy component. For each cluster k , the synergy matrix (Λ_k) is generated with cluster-specific bias patterns (B_k) that shift the activation balance among proximal, distal, or antagonistic muscle groups.

The parameter `cluster_sep_spatial` controls the degree of differentiation between clusters, ranging from 0 (identical synergy weights) to 2 (strongly distinct spatial modules).

Temporal Coordination Axis

The temporal axis governs how synergies are activated over time, including their onset timing, overlap, and rhythmic frequency. For each cluster, the activation matrix $C_i^k(t)$ is generated with cluster-specific frequencies (f_k) and phase offsets ($\phi_{ij}^k(k)$).

Three representative temporal coordination modes:

- Cluster 1: Co-contraction/rigid (all synergies in-phase, strong overlap)
- Cluster 2: Alternating/efficient (phases evenly distributed, minimal overlap)
- Cluster 3: Adaptive/reactive (random phases, heterogeneous frequencies)

The parameter `cluster_sep_temporal` (0-2) scales the magnitude of inter-cluster differences in phase and frequency.

Stability-Adaptability Axis

The stability axis introduces physiological and measurement variability into the EMG signal. Each cluster k is assigned distinct noise parameters to simulate differences in control precision and adaptability.

Three representative control profiles:

- Cluster 1: Stable control (minimal noise, elite or late rehab)
- Cluster 2: Moderate control (balanced smoothness vs flexibility)
- Cluster 3: Adaptive/exploratory (variable, early rehab or fatigue)

Examples

```
## Not run:
# Basic usage with default parameters
sim_v2 <- simulate_dynamic_synergy_data(
  N = 30, K = 3, r = 3, M = 8, T_each = 200
)

# High spatial separation, low temporal separation
```

```

sim_spatial <- simulate_dynamic_synergy_data(
  N = 30, K = 3, r = 3, M = 8, T_each = 200,
  cluster_sep_spatial = 2.0,
  cluster_sep_temporal = 0.5,
  cluster_sep_stability = 1.0,
  seed = 456
)

# Factorial design: 3x3x3 combinations
spatial_levels <- c(0.5, 1.0, 2.0)
temporal_levels <- c(0.5, 1.0, 2.0)
stability_levels <- c(0.5, 1.0, 2.0)

results <- list()
idx <- 1
for (sp in spatial_levels) {
  for (tp in temporal_levels) {
    for (st in stability_levels) {
      results[[idx]] <- simulate_dynamic_synergy_data(
        N = 30, K = 3, r = 3, M = 8, T_each = 200,
        cluster_sep_spatial = sp,
        cluster_sep_temporal = tp,
        cluster_sep_stability = st,
        seed = 100 + idx
      )
      idx <- idx + 1
    }
  }
}

## End(Not run)

```

simulation_comparison *Simulation Comparison Functions*

Description

Functions for comparing different methods (SingleFA, SinglePCA, MixtureFA, MixturePCA) on simulated data.

simulation_visualization

Simulation Visualization Functions

Description

Functions for visualizing simulation results, including ARI plots, BIC heatmaps, and VAF/reconstruction error visualizations.

strip_attributes	<i>Strip All Attributes Except Essential Ones</i>
------------------	---

Description

Removes all attributes from an object except for names, row.names, and class. This is useful for testing scenarios where you want to simulate data frames without custom attributes (e.g., fitted model objects stored as attributes).

Usage

```
strip_attributes(x)
```

Arguments

x	An R object (typically a data frame)
---	--------------------------------------

Value

The same object with all non-essential attributes removed

Examples

```
## Not run:
# Create a data frame with custom attributes
df <- data.frame(x = 1:3, y = 4:6)
attr(df, "custom_attr") <- "some_value"

# Strip custom attributes
df_clean <- strip_attributes(df)
# df_clean will have only names, row.names, and class attributes

## End(Not run)
```

submit_hpc_sweep	<i>Submit Parameter Sweep to HPC (Simple Interface)</i>
------------------	---

Description

A simplified wrapper for submitting parameter sweeps to HPC clusters. Automatically selects the appropriate parallelization strategy based on the specified cluster type.

Usage

```
submit_hpc_sweep(
  cluster_type = c("local", "pbs", "slurm", "sge"),
  queue = NULL,
  walltime = "04:00:00",
  memory = NULL,
  test = FALSE,
  cores = 4,
  output_dir = NULL,
  submit = TRUE,
  verbose = TRUE
)
```

Arguments

cluster_type	Character. Type of HPC cluster: • "local": Local multicore execution (no HPC) • "pbs": PBS/OpenPBS/TORQUE cluster • "slurm": SLURM cluster • "sge": SGE cluster
queue	Character. Queue/partition name for the cluster. Required for HPC clusters.
walltime	Character. Maximum wall time per job (default: "04:00:00").
memory	Character. Memory per job (default: "4gb" for PBS, "4096" for SLURM).
test	Logical. If TRUE, use test parameter grid. Default is FALSE.
cores	Integer. Number of cores for local execution. Default is 4.
output_dir	Character. Output directory for results.
submit	Logical. If TRUE, submit jobs. If FALSE, just generate scripts. Default is TRUE.
verbose	Logical. Print progress messages. Default is TRUE.

Details

This function provides a simplified interface for the most common HPC use cases. It automatically:

- Selects the appropriate future backend for the cluster type
- Configures resource specifications
- Handles PBS vs SLURM differences

For more control over the submission, use `run_param_sweep_modelsel` directly with `parallel_strategy = "future"` or `"pbs"`.

Value

Invisibly returns the result from `run_param_sweep_modelsel`.

See Also

[run_param_sweep_modelsel](#), [setup_future_backend](#)

Examples

```
## Not run:
# Local execution (8 cores)
submit_hpc_sweep("local", cores = 8, test = TRUE)

# PBS cluster
submit_hpc_sweep("pbs", queue = "batch", walltime = "02:00:00")

# SLURM cluster
submit_hpc_sweep("slurm", queue = "compute", memory = "8192")

# Dry run (generate scripts without submitting)
submit_hpc_sweep("pbs", queue = "batch", submit = FALSE)

## End(Not run)
```

submit_pbs_sweep

Submit PBS Parameter Sweep

Description

Prepares and optionally submits a PBS array job for parameter sweep.

Usage

```
submit_pbs_sweep(
  param_grid,
  output_dir,
  pbs_config = list(),
  sweep_type = c("modelsel", "baselines"),
  sweep_config = list(),
  submit = FALSE,
  verbose = TRUE
)
```

Arguments

param_grid	Data frame. Parameter grid to sweep.
output_dir	Character. Directory for output files.
pbs_config	List. PBS configuration options (see generate_pbs_script).
sweep_type	Character. Type of sweep: "modelsel" or "baselines".
sweep_config	List. Sweep-specific configuration.

submit	Logical. If TRUE, submit the job. If FALSE, only generate scripts.
verbose	Logical. Print progress messages.

Value

List containing:

- job_id: PBS job ID (if submitted)
- script_file: Path to the PBS script
- param_grid_file: Path to the saved parameter grid
- output_dir: Output directory
- n_tasks: Number of tasks

Examples

```
## Not run:
# Generate scripts only (dry run)
result <- submit_pbs_sweep(
  param_grid = make_param_grid_test(),
  output_dir = "results",
  submit = FALSE
)

# Submit job
result <- submit_pbs_sweep(
  param_grid = make_param_grid_paper_core(),
  output_dir = "results",
  pbs_config = list(queue = "batch"),
  submit = TRUE
)

## End(Not run)
```

summarize_vaf_sse_mfa *Summarize VAF and SSE for an MFA Model*

Description

Computes total SSE and VAF across all subjects, and optionally by cluster.

Usage

```
summarize_vaf_sse_mfa(mfa_fit, list_of_data, by_cluster = FALSE)
```

Arguments

mfa_fit	A fitted MFA model (with z, Lambda, Psi, mu, etc.).
list_of_data	The data list (a list of (T_i x M) matrices).
by_cluster	Logical; if TRUE, returns a data.frame with SSE/VAF per cluster.

Value

If by_cluster=FALSE, returns a named vector with SSE and VAF. If by_cluster=TRUE, returns a data.frame with columns Cluster, SSE, T, SSE_per_time, VAF.

```
summary.mixture_r_selection
```

Summary Method for mixture_r_selection Objects

Description

Summary Method for mixture_r_selection Objects

Usage

```
## S3 method for class 'mixture_r_selection'
summary(object, ...)
```

Arguments

object	A mixture_r_selection object.
...	Additional arguments (ignored).

```
summary.preprocessed_emg
```

Summary Method for preprocessed_emg Objects

Description

Summary Method for preprocessed_emg Objects

Usage

```
## S3 method for class 'preprocessed_emg'
summary(object, ...)
```

Arguments

object	A preprocessed_emg object
...	Additional arguments (ignored)

theme_mixsynergy	<i>Unified Theme and Style Definitions for synergyMixR Visualizations</i>
------------------	---

Description

This file provides consistent theme and style definitions for all visualization functions in the synergyMixR package, ensuring publication-quality graphics with unified aesthetics. Get Unified Theme for synergyMixR Plots

Usage

```
theme_mixsynergy(base_size = 11, base_family = "")
```

Arguments

- base_size Base font size (default: 11)
- base_family Base font family (default: "")

Details

Returns a consistent ggplot2 theme for all package visualizations. Based on theme_minimal() with customizations for publication quality.

Value

A ggplot2 theme object

Examples

```
## Not run:
library(ggplot2)
ggplot(data, aes(x, y)) + geom_point() + theme_mixsynergy()

## End(Not run)
```

theme_mixsynergy_heatmap	<i>Get Unified Heatmap Theme for synergyMixR</i>
--------------------------	--

Description

Returns a specialized theme for heatmap visualizations with minimal grid lines.

Usage

```
theme_mixsynergy_heatmap(base_size = 11, base_family = "")
```

Arguments

base_size	Base font size (default: 11)
base_family	Base font family (default: "")

Value

A ggplot2 theme object optimized for heatmaps

Examples

```
## Not run:
library(ggplot2)
ggplot(data, aes(x, y, fill = value)) +
  geom_tile() +
  theme_mixsynergy_heatmap()

## End(Not run)
```

theme_sweep	<i>Theme for Parameter Sweep Plots</i>
-------------	--

Description

A consistent theme for parameter sweep visualization that ensures readable text, clear facets, and prevents layout collapse.

Usage

```
theme_sweep(base_size = 12, base_family = "")
```

Arguments

base_size	Base font size. Default is 12.
base_family	Base font family. Default is "".

Value

A ggplot2 theme object

Examples

```
## Not run:
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  theme_sweep()

## End(Not run)
```

visualize_twostep_diagnostics
<i>Visualization Functions for Two-Step Algorithm Diagnostics</i>

Description

This file contains visualization functions to diagnose failures in the Two-step algorithm (fit_two_step_baseline). These functions help identify issues with basis alignment, clustering, and reconstruction.

wait_for_pbs_job	<i>Wait for PBS Job Completion</i>
------------------	------------------------------------

Description

Blocks until a PBS job completes.

Usage

```
wait_for_pbs_job(  
  job_id,  
  check_interval = 60,  
  timeout = NULL,  
  initial_wait = 5,  
  output_dir = NULL,  
  n_tasks = NULL,  
  verbose = TRUE  
)
```

Arguments

job_id	Character. PBS job ID.
check_interval	Integer. Seconds between status checks.
timeout	Integer. Maximum seconds to wait (NULL for no timeout).
initial_wait	Integer. Seconds to wait before first status check. This gives time for the job to be registered in the queue.
output_dir	Character. If provided, also monitors partial_results directory to track progress.
n_tasks	Integer. Total number of expected tasks (used with output_dir).
verbose	Logical. Print progress messages.

Value

Logical. TRUE if job completed, FALSE if timeout.

Examples

```
## Not run:  
wait_for_pbs_job("12345", timeout = 3600) # Wait up to 1 hour  
  
## End(Not run)
```

Index

aggregate_partial_results, [5](#), [134](#)
align_clusters_to_truth, [6](#), [34](#)
align_factor_signs_mfa, [7](#), [8](#)
align_factor_signs_mpca, [8](#)
assign_by_kmeans, [9](#)

bandpass_filter, [10](#), [120](#)
bootstrap_loading_ci, [11](#)

calc_reconstruction_error_mixtureFA, [13](#)
calc_reconstruction_error_mixturePCA, [14](#)
calc_reconstruction_error_singleFA, [14](#)
calc_reconstruction_error_singlePCA, [15](#)
calc_total_SST, [15](#)
calibrate_model, [16](#)
check_future_available, [16](#)
check_pbs_job_status, [17](#)
check_SSE_diagnostics, [17](#)
compare_baselines, [20](#), [134](#)
compare_static_vs_dynamic, [20](#), [22](#), [23](#)
compute_ari, [26](#)
compute_ari_vaf_table, [27](#)
compute_BIC, [27](#)
compute_BIC_mfa, [28](#), [28](#), [45](#)
compute_BIC_mpca, [28](#), [29](#), [45](#)
compute_bic_wrapper, [30](#)
compute_calibration, [31](#), [35](#), [95](#)
compute_cluster_sizes, [32](#), [33](#), [116](#)
compute_cluster_sizes_mpca, [33](#)
compute_clusterwise_subspace_score, [33](#)
compute_ece, [32](#), [35](#)
compute_entropy, [36](#), [45](#)
compute_factor_scores, [37](#), [42](#), [117](#)
compute_factor_scores_mfa, [38](#)
compute_factor_scores_mpca, [39](#)
compute_factor_scores_mpca_single, [39](#), [40](#)

compute_factor_scores_optimized, [38](#), [40](#)
compute_global_vaf_mfa, [37](#), [41](#), [42](#), [52](#), [116](#)
compute_global_vaf_mpca, [42](#), [52](#)
compute_ICL, [43](#)
compute_ICL_mfa, [44](#), [44](#)
compute_ICL_mpca, [44](#), [45](#)
compute_logLik, [46](#)
compute_logLik_mfa, [28](#), [46](#), [47](#)
compute_logLik_mpca, [29](#), [46](#), [47](#)
compute_mvc_reference, [48](#)
compute_reconstruction_error, [49](#)
compute_smoothness_penalty, [50](#)
compute_subspace_similarity, [51](#)
compute_VAF, [52](#)
compute_VAF_mfa, [53](#)
compute_VAF_mpca, [54](#)
compute_vaf_wrapper, [54](#)
compute_vss_criterion, [55](#)
count_completed_tasks, [57](#)

emg_preset_config, [57](#), [120](#)
evaluate_model_selection, [58](#)
extract_cluster_labels, [59](#)
extract_envelope, [60](#), [120](#)
extract_selection_summary, [62](#)
extract_subject_features_by_singlePCA, [62](#)

facet_mixsynergy, [63](#)
fit_single_factor_analysis, [64](#)
fit_single_pca, [65](#)
fit_two_step_baseline, [65](#)

generate_pbs_script, [67](#), [193](#)
generate_synergy_data, [68](#)
get_model_by_K_r, [71](#)
get_model_by_K_r_mpca, [72](#)

make_table1, [72](#)

make_table2, 73
 mfa_em_fit, 12, 73, 79
 mfa_em_fit_cpp_singleInit, 75, 76
 mixture_pca_em_fit, 12, 43, 75, 77
 mixture_pca_em_fit_cpp_singleInit, 80

 normalize_emg, 81, 120
 notch_filter, 83, 120

 plot.mixture_r_selection, 84
 plot_activation_patterns, 85
 plot_aligned_basis_heatmaps, 86
 plot_all_factor_scores_mfa, 87
 plot_all_factor_scores_mPCA, 88
 plot_all_reconstructed_waveforms_mfa, 89
 plot_all_reconstructed_waveforms_mPCA, 90
 plot_basis_umap, 91
 plot_bic_comparison, 93
 plot_bic_line, 94
 plot_calibration, 32, 94
 plot_cluster_embedding, 96
 plot_cluster_synergy_loadings_mfa, 75, 97
 plot_cluster_synergy_loadings_mPCA, 79, 98
 plot_confusion_matrix, 99
 plot_fig1_ARI, 100
 plot_fig1_baseline_comparison, 101
 plot_fig2_BIC_heatmap, 102
 plot_fig3_init_boxplot, 103
 plot_fig4to6_reconstruction_vaf, 103
 plot_metric_vs_axis, 104
 plot_neural_command, 105
 plot_noise_profiles, 106
 plot_reconstructed_emg, 107
 plot_sse_heatmap, 108
 plot_sweep_summary, 109, 134
 plot_synergy_bases, 110
 plot_synergy_correlation_comparison, 111
 plot_synergy_summary, 113
 plot_waveform_comparison_mfa, 114
 plot_waveform_comparison_mPCA, 114
 plot_winrate_heatmap, 115
 posthoc_mfa_evaluation, 116
 posthoc_mPCA_evaluation, 117
 preprocess_emg, 118

 print.bootstrap_ci, 120
 print.calibration_result, 121
 print.mixture_r_selection, 121
 print.preprocessed_emg, 122

 read_all_results, 27, 122
 read_and_visualize_all, 123
 readRDS, 122
 reconstruct_singleMPCA, 124
 rectify, 120, 124
 remove_dc_offset, 125
 rotate_mfa_model, 7, 126
 rotate_mPCA_model, 127
 run_k_selection_comparison, 128
 run_param_sweep, 22, 129, 133, 134
 run_param_sweep_baselines, 131
 run_param_sweep_modelsel, 70, 135, 192, 193
 run_single_task, 139
 run_sweep_future, 141

 scale_color_mixsynergy, 142
 scale_fill_mixsynergy, 143
 scale_fill_mixsynergy_discrete, 143
 scale_fill_mixsynergy_diverging, 144
 select_K_bootstrap_stability, 145
 select_K_crossvalidation, 146
 select_K_r, 58, 62, 147
 select_optimal_K_r_mfa, 71, 116, 117, 149, 150
 select_optimal_K_r_mfa_twostage, 149, 151
 select_optimal_K_r_mPCA, 72, 117, 149, 153
 select_optimal_K_r_mPCA_twostage, 149, 155
 select_r, 75, 79
 select_r_by_smoothness_mfa, 157
 select_r_by_smoothness_mPCA, 158
 select_r_by_vaf_mfa, 160
 select_r_by_vaf_mPCA, 161
 select_r_by_vss_mfa, 162
 select_r_by_vss_mPCA, 164
 select_r_clusterwise_mfa, 166, 172
 select_r_clusterwise_mPCA, 170
 select_r_global_mfa, 173
 select_r_global_mPCA, 175
 select_r_mixture, 178
 select_r_multicriteria_mfa, 181

`select_r_multicriteria_mPCA`, 183
`setup_future_backend`, 133, 137, 141, 185, 193
`show_parallel_options`, 187
`simulate_dynamic_synergy_data`, 22–24, 68–70, 187
`simulation_comparison`, 190
`simulation_visualization`, 190
`strip_attributes`, 191
`submit_hpc_sweep`, 191
`submit_pbs_sweep`, 133, 137, 193
`summarize_vaf_sse_mfa`, 194
`summary.mixture_r_selection`, 195
`summary.preprocessed_emg`, 195

`theme_mixsynergy`, 196
`theme_mixsynergy_heatmap`, 196
`theme_sweep`, 197

`visualize_twostep_diagnostics`, 198

`wait_for_pbs_job`, 198