Faculty of Mathematics and Information Sciences

WARSAW UNIVERSITY OF TECHNOLOGY

Deliverable 5

Post-completion Documentation

Adapting the gips library for classification problem utilizing discriminant analysis – gipsDA

Authors:
Norbert Frydrysiak
Antoni Kingston

Supervisors:
MSc Eng. Adam Chojecki
PhD Bartosz Kołodziejek, Assoc. Prof.

Version 1.0

January 6, 2026

Contents

1 Abstract

This document serves as the final post-completion report for the `gipsDA` project, an R library designed to enhance discriminant analysis through permutation symmetry constraints. It acts as the definitive technical guide for the deployment, installation, and operational utilization of the software.

The report details the system configuration verified on high-performance architectures (Apple Silicon and x86_64) and provides a comprehensive installation guide for both remote and local environments. It presents the results of acceptance testing against the initial functional and non-functional requirements, highlighting a robust quality assurance process that achieved approximately 80% code coverage using the `testthat` framework. Furthermore, the document outlines significant technical extensions introduced during the final phase, including a novel spectral decomposition method for synthetic data generation. A detailed User's Manual is provided to facilitate immediate adoption by Data Scientists, followed by a retrospective analysis of the engineering experience gained during the project's lifecycle.

2 History of changes

Authors	**Description**	**Version**	**Date**
Norbert Frydrysiak	Initial structure of the Document.	0.1	28.12.2025
Antoni Kingston	Added Deployment and Installation sections.	0.2	30.12.2025
Norbert Frydrysiak	Added Acceptance Tests and User Manual.	0.5	02.01.2026
Antoni, Norbert	Finalized Project Experience and formatting.	1.0	03.01.2026

3 References to Previous Deliverables

During the final implementation and validation phases, specific advancements were made relative to the initial design outlined in previous deliverables. These changes were driven by the need for higher software reliability and more rigorous simulation scenarios.

3.1 Quality Assurance and Testing Framework

A major architectural decision was the complete migration of the testing suite to the `testthat` framework, the current industry standard for R package development. This transition facilitated a more structured and automated testing process.

We achieved a **code coverage of approximately 80%**, meticulously verifying the critical components of the package. The testing strategy prioritizes:

- **Algorithmic Core:** Verifying the mathematical correctness of the `gips` backend integration and permutation optimization.
- **S3 Method Dispatch:** Ensuring seamless integration with standard R generics like `print` and `predict`.
- **Edge Cases:** Handling degenerate cases, such as zero-variance features or single-class inputs, to ensure production-grade stability.

3.2 Enhanced Synthetic Data Generation

To rigorously stress-test the models under varied spectral conditions, we introduced an additional method for generating synthetic covariance matrices. Unlike the standard Wishart-based

approach, this method constructs positive semi-definite (PSD) matrices via **spectral decomposition**. This allows for direct control over the distribution of eigenvalues (and thus the condition number of the matrix).

The algorithm, implemented as `generate_lamb_q`, follows this mathematical procedure:

1. **Eigenvalue Generation:** A vector of eigenvalues $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$ is drawn from a specified distribution $\mathcal{D}$ (e.g., Exponential) and sorted such that $\lambda_1 \geq \cdots \geq \lambda_p > 0$. The diagonal matrix $\boldsymbol{\Lambda}$ is defined as:

$$\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_p) \tag{1}$$

2. **Random Orthogonal Matrix Generation:** A random matrix $\mathbf{Z} \in \mathbb{R}^{p \times p}$ is generated with entries sampled from a standard normal distribution:

$$Z_{ij} \sim \mathcal{N}(0, 1) \tag{2}$$

   To obtain a uniformly distributed orthogonal matrix (Haar measure), we perform a QR decomposition on $\mathbf{Z}$:

$$\mathbf{Z} = \mathbf{Q}'\mathbf{R} \tag{3}$$

   The matrix $\mathbf{Q}'$ is then adjusted by the signs of the diagonal elements of $\mathbf{R}$ to ensure uniqueness and correct distribution:

$$\mathbf{Q} = \mathbf{Q}' \cdot \operatorname{diag}\left(\operatorname{sgn}(R_{11}), \ldots, \operatorname{sgn}(R_{pp})\right) \tag{4}$$

3. **Covariance Matrix Construction:** The final covariance matrix $\boldsymbol{\Sigma}$ is constructed via spectral reconstruction:

$$\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top \tag{5}$$

This methodology ensures that the generated matrices are strictly positive semi-definite while possessing a randomized rotation and a controlled eigenstructure.

4 Deployment Documentation

4.1 System Configuration

The `gipsDA` library is architected to be fully **platform-agnostic**. It operates within the R statistical computing environment, abstracting away the underlying hardware and operating system differences. Consequently, the package can be deployed on any system: UNIX-based or Windows that supports a compatible version of R.

R Environment

- **Verified Version:** The codebase was developed and rigorously tested using **R version 4.5.1**.
- **Compatibility:** While 4.5.1 is the reference version, the package is expected to maintain backward compatibility with the R 4.x series.

Verified Reference Platforms

To ensure robustness across different architectures and operating systems, the deployment was verified on the following specific configurations:

- **Apple Silicon (ARM64):**
  - **Hardware:** MacBook Pro with **Apple M4 Max** processor (14-core).
  - **OS:** macOS (Tahoe 26.1).
  - **Status:** Fully Verified.
- **Linux (x86_64):**
  - **Hardware:** Workstation with **AMD Ryzen 7 8840U** processor.
  - **OS:** Arch Linux distribution (Kernel 6.17).
  - **Status:** Fully Verified.
- **Windows (x86_64):**
  - **Hardware:** Standard Workstation / PC.
  - **OS:** Microsoft Windows 10.
  - **Status:** Fully Verified.

Windows Toolchain

For Windows users installing the package from source (e.g., via GitHub), we recommend ensuring that **Rtools** is installed on the target machine. This toolchain facilitates the compilation of source packages and ensures smooth handling of dependencies.

4.2 Dependencies

To ensure the correct functioning of the package, several external R libraries are required. These are listed in the `Imports` field of the package description and are automatically handled during the installation process. The table below lists the minimum versions verified during the final validation phase.

Package	Min. Version	Role
`gips`	$\geq$ 1.2.3	Core backend for permutation symmetry discovery.
`jsonlite`	$\geq$ 2.0.0	Handles model serialization (JSON export/import).
`permutations`	$\geq$ 1.1-6	Provides mathematical operations on permutations.
`dplyr`	$\geq$ 1.1.4	Data manipulation pipelines.
`tidyr`	$\geq$ 1.3.2	Data tidying and cleaning tools.
`tibble`	$\geq$ 3.3.0	Modern data frame handling.
`rlang`	$\geq$ 1.1.6	Tidy evaluation backend.
`ggplot2`	$\geq$ 4.0.1	Primary engine for visualizing model results.
`lattice`	$\geq$ 0.22-7	Auxiliary plotting tools.
`patchwork`	$\geq$ 1.3.2	Tool for composing multiple plots.

Table 1: Runtime Dependencies (Imports)

Development and Testing

For development purposes, running the test suite, or building documentation, the following additional packages are suggested or required:

- **Unit Testing:** The project utilizes the `testthat` framework ($\geq$ 3.3.1) along with `mockery` ($\geq$ 0.4.5) for mocking objects to ensure code reliability.
- **Comparison & Baselines:** The `MASS` library ($\geq$ 7.3-65) is suggested for providing baseline LDA/QDA implementations for comparative analysis.
- **Documentation:** `roxygen2` ($\geq$ 7.3.3) is used for generating documentation from source comments.

5 Installation Instruction

The `gipsDA` package is currently hosted on a GitHub repository. Users can choose between installing the latest version directly from the remote source or building the package manually from a local clone.

5.1 Prerequisites

Ensure that a compatible version of R is installed. To install packages directly from GitHub, we recommend using the lightweight `remotes` package. Open an R console and execute:

Listing 1: R Console

```
if (!require("remotes")) install.packages("remotes")
```

5.2 Option A: Remote Installation (Recommended)

This is the most straightforward method to get the latest development version of the package. Run the following command in your R console:

Listing 2: R Console

```r
# Download and install directly from GitHub using remotes
remotes::install_github("AntoniKingston/gipsDA")

# Note: If you prefer the 'devtools' suite, it wraps 'remotes'
    functionality,
# so devtools::install_github("AntoniKingston/gipsDA") will also work.
```

5.3 Option B: Local Installation from Source

If you prefer to inspect the code before installation or need to work offline with downloaded sources, follow these steps:

1. **Clone the repository** using your terminal/command prompt:

```
>_ Terminal

git clone https://github.com/AntoniKingston/gipsDA.git
```

2. **Install from the local directory** using R. Navigate to the parent directory of the cloned folder in your R console and run:

Listing 3: R Console

```r
# Install from the local source directory
install.packages("./gipsDA", repos = NULL, type = "source")
```

5.4 Future Availability (CRAN)

We plan to submit the `gipsDA` package to the Comprehensive R Archive Network (CRAN) in the near future. Once accepted, the installation process will be standardized, allowing users to install it without additional dependencies like `remotes`:

Listing 4: R Console

```r
# Planned future installation method (Standard CRAN)
install.packages("gipsDA")
```

5.5 Verification

To verify that the installation was successful and the library is correctly registered in your environment, you can check the list of installed packages:

Listing 5: R Console

```r
# Check if gipsDA is present in the installed packages list
if ("gipsDA" %in% rownames(installed.packages())) {
    message("gipsDA is successfully installed and ready to use.")
} else {
    stop("Installation failed: gipsDA not found.")
}
```

6 Acceptance Tests

The following table summarizes the validation of the system against the Functional and Non-Functional Requirements defined in Deliverable 1. The results are based on the comprehensive testing reported in Deliverable 4.

ID	Requirement	Validation Result (Del. 4)	Status
Functional Requirements			
F1	**Train Classifier:** User can train a model using `fit()` equivalent.	Validated via `gipsLDA`, `gipsQDA`, and `gipsMultQDA` functions on synthetic and real data.	**PASSED**
F2	**Generate Predictions:** User can predict classes on new data.	Validated via `predict()` method. Probabilities sum to 1; verified on Iris dataset.	**PASSED**
F3	**Inspect Internals:** User can access learned parameters (matrices, symmetries).	Validated via `print()` methods showing "Found Permutation" and optimization info.	**PASSED**
F4	**Serialize Model:** Save/Load model for deployment.	Validated via `gipsDA_to_json` and `gipsDA_from_json`. Full round-trip integrity confirmed.	**PASSED**
Non-Functional Requirements			
NF1	**API Consistency:** Interface must match standard R (`MASS`).	The package uses S3 methods (`formula`, `predict`) identical to `MASS::lda`.	**PASSED**
NF2	**Documentation:** Full `roxygen2` documentation.	All exported functions have help pages accessible via `?function`.	**PASSED**
NF3	**Error Messaging:** Informative errors for invalid input.	"Negative testing" confirmed human-readable errors (e.g., dimension mismatch).	**PASSED**
NF4	**High Test Coverage:** Comprehensive unit tests.	Implemented using `testthat`. Covers backend, interface, and edge cases.	**PASSED**
NF5	**Performance:** Reasonable training time on consumer hardware.	Verified on M4 Max and Ryzen 7. Immediate results ($\approx$ 2s) for $p \leq 7$ (BF).	**PASSED**
NF6	**Deployment:** Minimal dependencies.	Depends only on stable CRAN packages (`gips`, `MASS`, `jsonlite`).	**PASSED**

7 User's Manual

This section provides a comprehensive guide for using the `gipsDA` package. The library is designed as an extension to the popular `MASS` package, preserving the standard R model-building interface (formula syntax) while injecting the novel covariance estimation methodology.

7.1 Loading the Library

To begin, load the package into your R session:

Listing 6: R Console

```
library(gipsDA)
```

7.2 Model Configuration & Hyperparameters

Before training specific models, it is important to understand the key hyperparameters available across all functions (`gipsLDA`, `gipsQDA`, `gipsMultQDA`). These parameters control how the permutation symmetry is discovered and applied.

- `MAP` **(Logical):** Controls the estimation strategy.
  - `TRUE` (Default): Uses the *Maximum A Posteriori* (argmax) approach. The covariance matrix is projected onto the single most probable permutation group found.
  - `FALSE`: Uses the *Bayesian Model Averaging* approach. The estimator is calculated as a weighted average of covariance matrices over the posterior distribution of permutation groups.
- `optimizer` **(Character):** Specifies the search algorithm for finding the optimal permutation group.
  - `NULL` (Default): Automatically selects the algorithm based on dimensionality. For small datasets ($p \leq 9$), it uses `"BF"` (Brute Force). For larger datasets ($p > 9$), it switches to `"MH"` (Metropolis-Hastings).
  - `"BF"`: Forces an exhaustive Brute Force search (guarantees optimal solution, but slow for $p > 9$).
  - `"MH"`: Forces the stochastic Metropolis-Hastings search.
- `max_iter` **(Integer):** Defaults to `100`. Specifies the number of iterations for the Metropolis-Hastings algorithm. This parameter is ignored if the optimizer is set to `"BF"`.

7.3 Training Models

7.3.1 gipsLDA (Linear Discriminant Analysis)

Use this function when you assume homoscedasticity (all classes share a common covariance matrix). It extends `MASS::lda`.

Listing 7: Training gipsLDA

```
# Train using the standard formula interface
# weighted_avg = FALSE corresponds to the standard pooled estimator
fit_lda <- gipslda(Species ~ ., data = iris, weighted_avg = FALSE)

# Print model summary (shows found symmetries)
print(fit_lda)
```

Console Output

```
Prior probabilities of groups:
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Group means:
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026

Coefficients of linear discriminants:
                    LD1        LD2
Sepal.Length  0.8635841 -0.1298911
Sepal.Width   1.5030221 -2.0678118
Petal.Length -2.2124913  1.0417746
Petal.Width  -2.7306422 -2.9735718

Proportion of trace:
  LD1   LD2
0.991 0.009

Permutations with their estimated probabilities:
[1] (13)(24)
```

7.3.2 gipsQDA (Quadratic Discriminant Analysis)

Use this for heteroscedastic assumptions (each class has its own covariance matrix). This is the most flexible model, where `gips` is applied independently to each class.

Listing 8: Training gipsQDA

```
# MAP = FALSE uses the weighted posterior estimator
fit_qda <- gipsqda(Species ~ ., data = iris, MAP = FALSE)

print(fit_qda)
```

Console Output

```
Call:
gipsqda(Species ~ ., data = iris, MAP = FALSE)

Prior probabilities of groups:
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Group means:
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026

Permutations with their estimated probabilities:
(1,3)(2,4)         ()      (2,4)      (1,3)
0.69108586 0.17759118 0.10792401 0.02339895
```

7.3.3 gipsMultQDA (Multi-Group Shared Symmetry)

This intermediate model assumes that while classes have different covariance matrices, they share the same underlying permutation symmetry structure.

Listing 9: Training gipsMultQDA

```
# Explicitly setting the optimizer for higher dimensions
fit_mult <- gipsmultqda(Species ~ ., data = iris,
                        optimizer = "MH", max_iter = 1000)
```

7.4 Making Predictions

Once a model is trained, use the standard `predict()` method to classify new observations.

Listing 10: Prediction

```
# Predict on new data
preds <- predict(fit_lda, iris)

# Access class labels
head(preds$class)

# Access posterior probabilities
head(preds$posterior)
```

Console Output

```
> head(preds$class)
[1] setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica
> head(preds$posterior)
  setosa   versicolor    virginica
1      1 1.134638e-21 2.576414e-41
2      1 1.706757e-17 3.820743e-36
3      1 4.137273e-19 4.370757e-38
4      1 3.488968e-16 3.062100e-34
5      1 5.205922e-22 1.171863e-41
6      1 1.000738e-20 3.698655e-39
```

7.5 Model Serialization (Saving/Loading)

For production deployment, models can be saved to JSON files using the custom serialization engine provided by the package. This ensures that the complex permutation objects are correctly preserved.

Listing 11: Serialization

```
# Save the trained model to a file
gipsDA_to_json(fit_lda, "model_lda.json")

# ... later, in a fresh session ...
loaded_model <- gipsDA_from_json("model_lda.json", "gipslda")

# The loaded model is ready for prediction immediately
predict(loaded_model, iris[1:5, ])
```

8 Project Experience Documentation

8.1 Technical Knowledge Acquired

- **Advanced R Development:** We gained deep expertise in the S3 object-oriented system, package structure, and namespace management. Specifically, extending the functionality of the established `MASS` library required a thorough understanding of method dispatch and environment scoping.
- **Mathematical Optimization:** Implementing the `gipsmult` backend provided practical experience with Bayesian statistics, specifically Metropolis-Hastings algorithms and permutation group theory.
- **Serialization Challenges:** We learned that standard serialization (pickling/RDS) is not always sufficient for interoperability. Developing a custom JSON engine required mapping complex R objects (formulas, attributes) to text-based formats and back.
- **Version Control Proficiency:** We significantly advanced our skills in using Git and GitHub for collaborative software development. Managing a complex codebase required strict adherence to branching strategies, regular pull requests, and code reviews. We learned to effectively resolve merge conflicts and maintain a clean project history, ensuring a stable development lifecycle.

8.2 Lessons Learned

- **The Importance of "Negative Testing":** Initially, we focused on proving the models work. However, during the Usability Analysis (Deliverable 4), we realized that robust error handling for invalid inputs is just as critical for the end-user experience (Data Scientist actor).
- **Small Data Regularization:** The empirical results confirmed our hypothesis: imposing symmetry constraints acts as a powerful regularizer. We observed that in low-sample regimes, `gipsDA` models significantly outperform standard QDA, validating the core business goal of the thesis.
- **Documentation as a Feature:** Writing documentation (Deliverable 1, 2, 3) in parallel with code development ensured that the final API was consistent and intuitive. It prevented "feature creep" by keeping us aligned with the initial Functional Requirements.
- **Collaborative Workflow:** This project highlighted that successful software engineering extends beyond writing code. We learned to effectively divide responsibilities—balancing the backend mathematical logic with the user-facing API implementation—and synchronized our efforts through constant communication. This experience improved our ability to work as a cohesive unit, mitigating the friction often found in parallel development streams.