# Jaya R Package — Updated Features & Use Cases (with Results)

Kartik Kasrewar

2025-08-26

## Contents



This vignette demonstrates the updated **Jaya** functions and **prints results and plots inline**. It sources the stand-alone R scripts (`jaya.R`, `jaya_multi.R`, `plotting.R`, `summary.R`). When packaged, these functions will be exported and S3 methods registered automatically.

## 1. Prerequisites

```r
# install.packages(c("ggplot2", "GGally", "future", "future.apply"), dependencies = TRUE)

source("jaya.R")
source("jaya_multi.R")
source("plotting.R")
source("summary.R")

has_gg <- requireNamespace("ggplot2", quietly = TRUE)
has_ggally <- requireNamespace("GGally", quietly = TRUE)
```

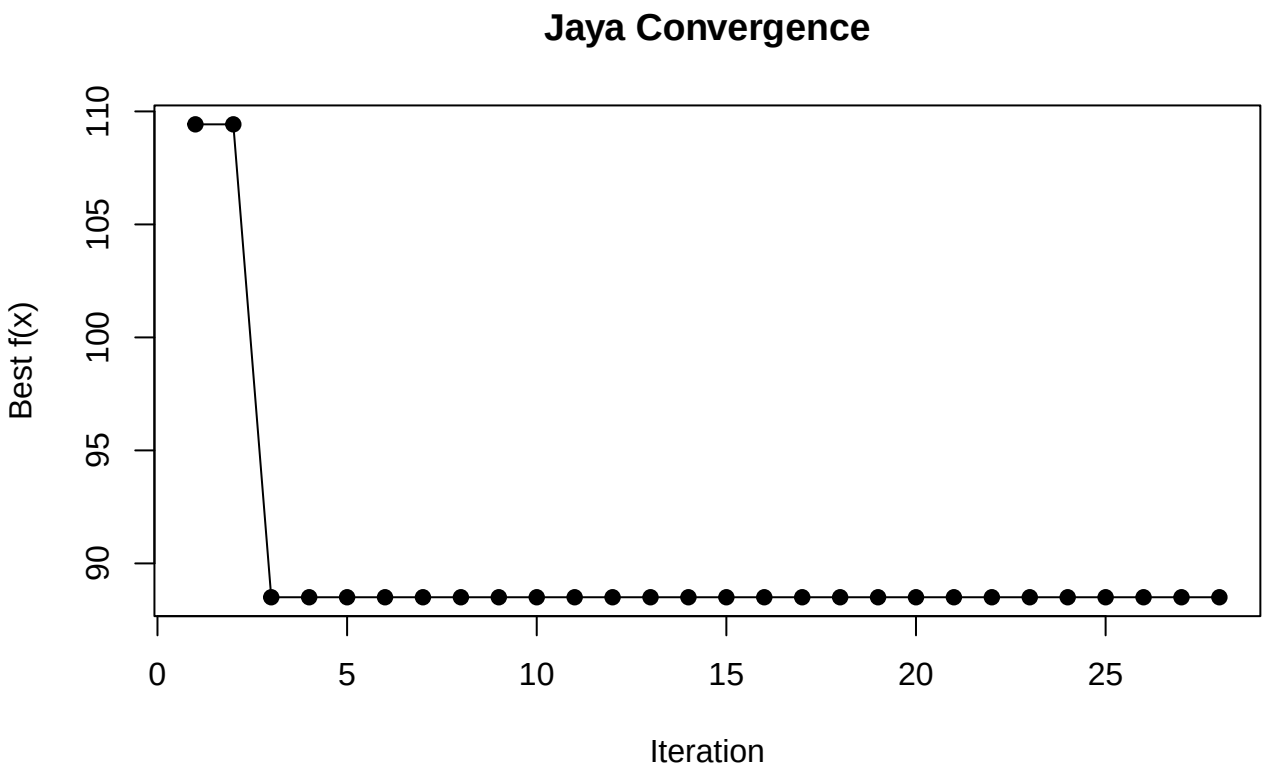
---

## 2. Single-Objective — Rastrigin (unconstrained) with Plots and Best Solution

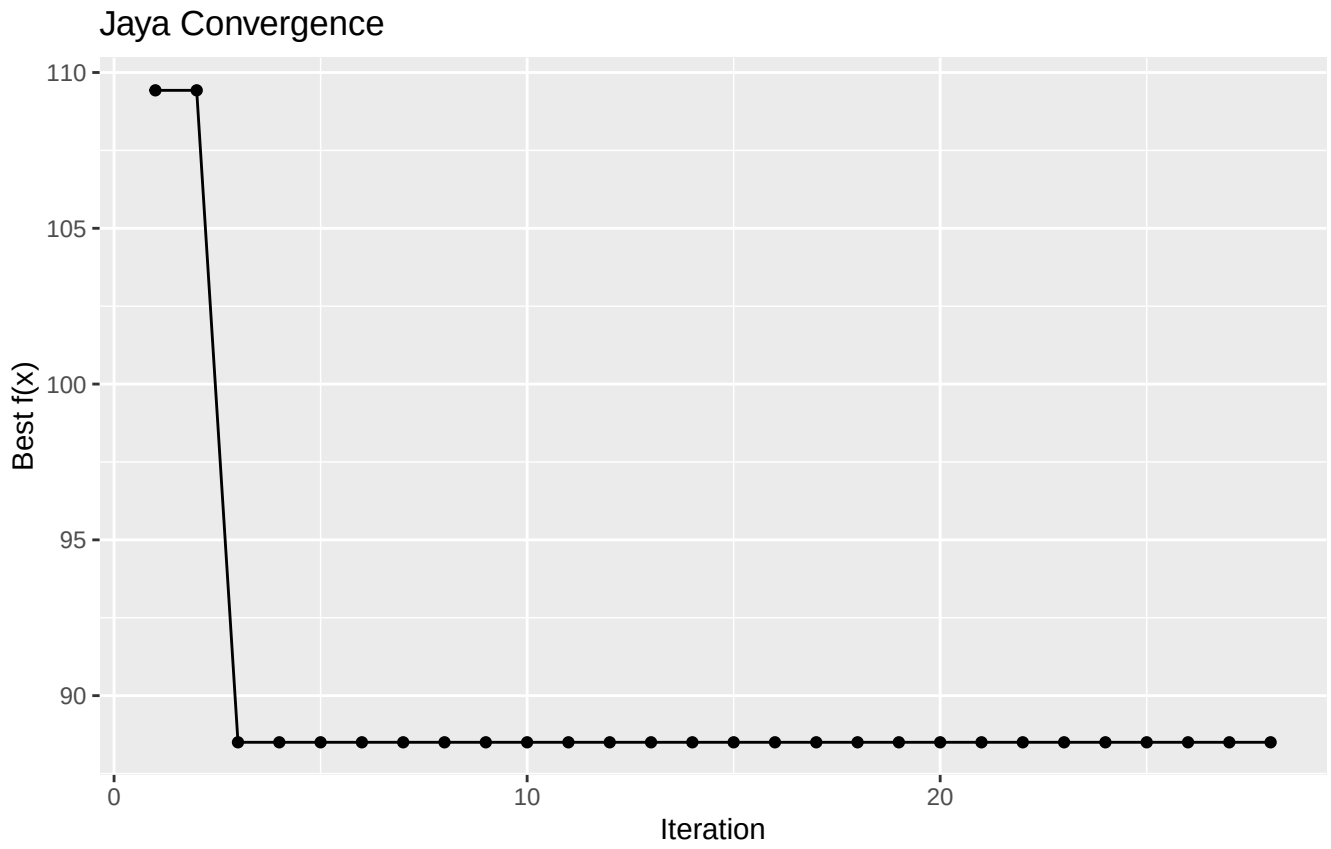
```r
rastrigin <- function(x) 10*length(x) + sum(x^2 - 10*cos(2*pi*x))

set.seed(42)
fit <- jaya(fun = rastrigin,
            lower = rep(-5.12, 10),
            upper = rep( 5.12, 10),
            popSize = 80, maxiter = 300,
            opt = "minimize",
            verbose = TRUE)

summary(fit)
```

```
## Jaya (single-objective) summary
## Variables: 10 | popSize: 80 | maxiter: 300
## Best value: 88.50279952
## Best parameters:
##          x1          x2          x3          x4          x5          x6
##  1.17621457  2.07384433 -4.33104723  0.97076350  3.85744479  1.10003473
##          x7          x8          x9         x10
## -0.14953113 -1.98261685 -0.02856725  2.86816288
## Evaluations: 2320 | Runtime: 0.020 s
```

```r
# Base plot
plot(fit)
```




```
# ggplot version (if available)
if (has_gg) {
  p <- autoplot.jaya(fit)
  print(p)
}
```




```
# Show best parameters and value
fit$par
```

```
##  [1]  1.17621457  2.07384433 -4.33104723  0.97076350  3.85744479  1.10003473
##  [7] -0.14953113 -1.98261685 -0.02856725  2.86816288
```

```
fit$value
```

```
## [1] 88.5028
```

---

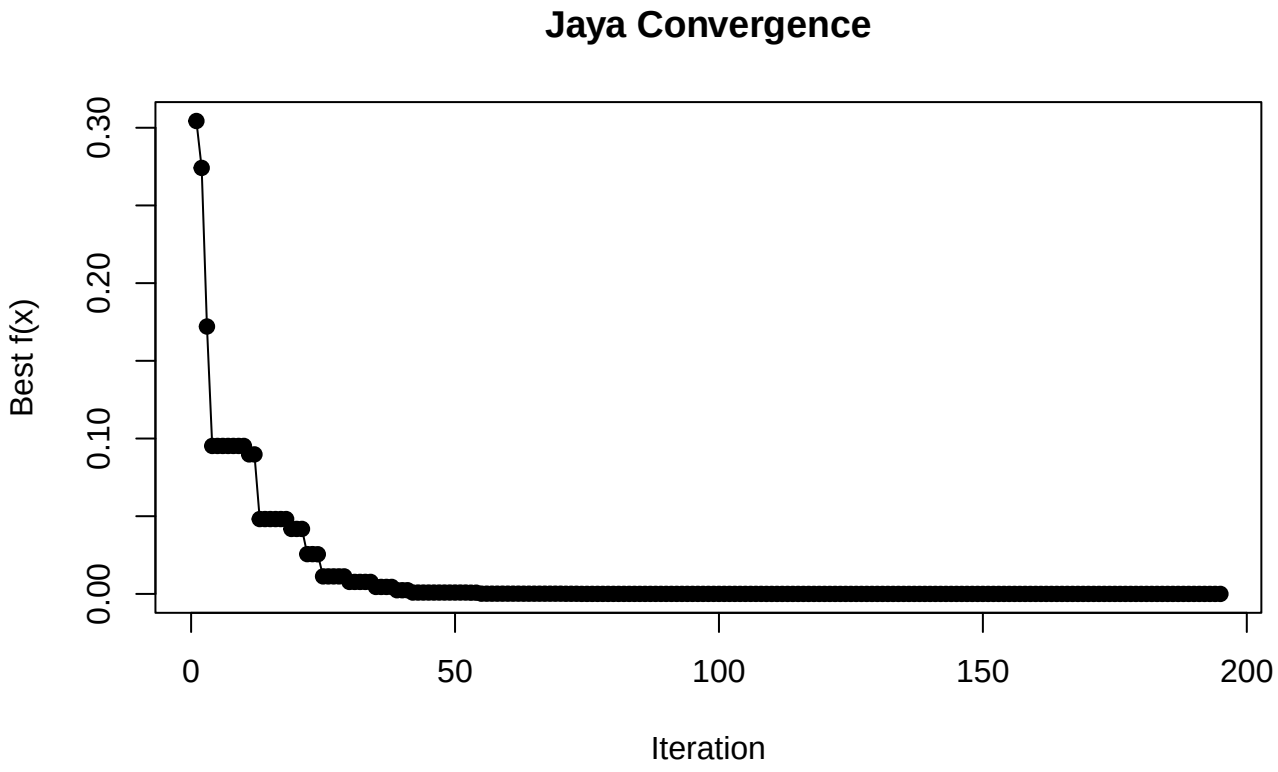
### 3. Single-Objective with Constraints — Reject vs Penalty (Results Shown)

```
sphere <- function(x) sum(x^2)
g1 <- function(x) sum(x) - 2  # <= 0 feasible

# (A) Reject infeasible candidates
set.seed(1)
fit_reject <- jaya(fun = sphere,
                   lower = rep(-1, 5), upper = rep(1, 5),
                   constraints = list(g1),
```

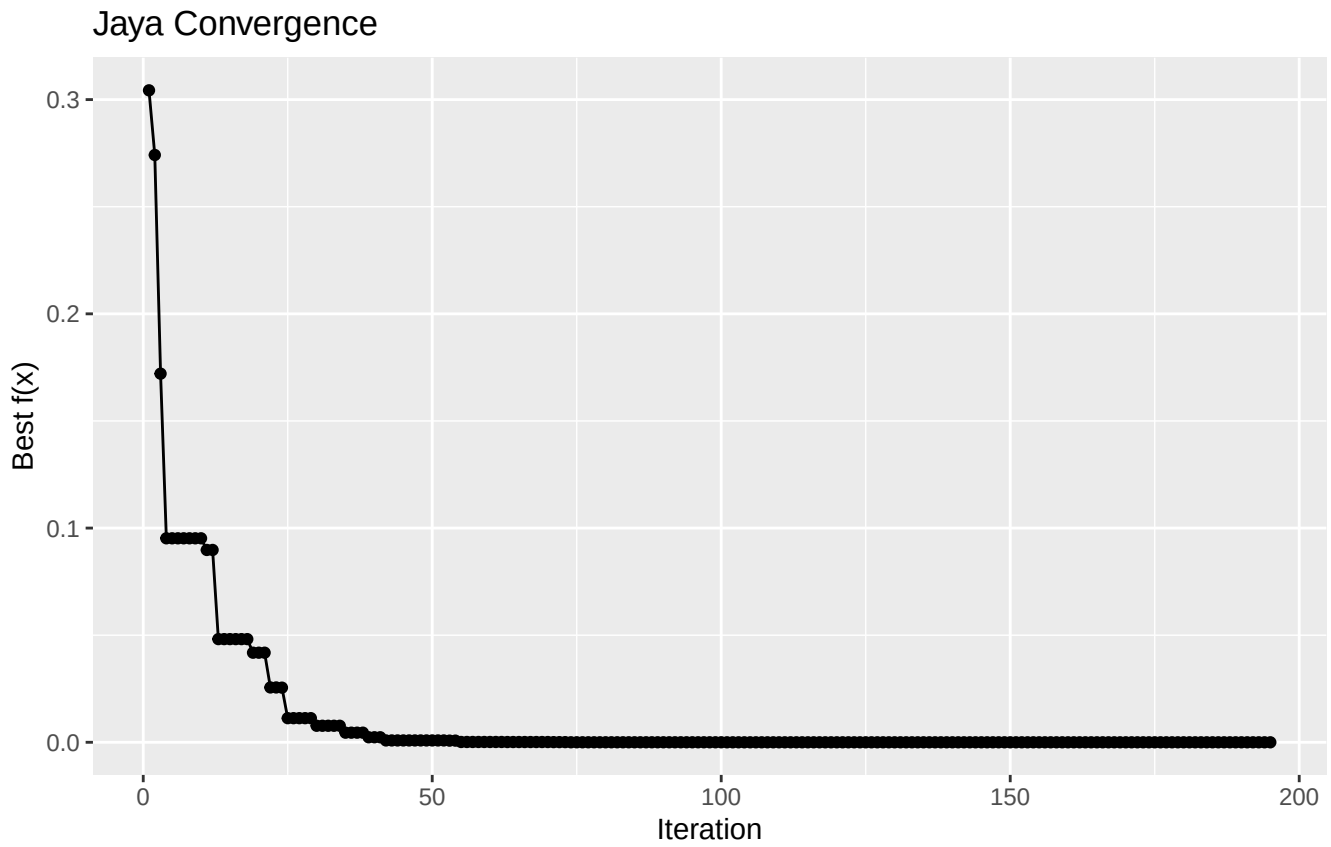
```
                  constraint_handling = "reject",
                  popSize = 50, maxiter = 200)
summary(fit_reject)
```

```
## Jaya (single-objective) summary
## Variables: 5 | popSize: 50 | maxiter: 200
## Best value: 1.377593539e-09
## Best parameters:
##            x1            x2            x3            x4            x5
##  1.636431e-05  9.454442e-07  1.304584e-06  1.073929e-06 -2.730374e-06
## Evaluations: 9805 | Runtime: 0.060 s
```

```
plot(fit_reject)
```



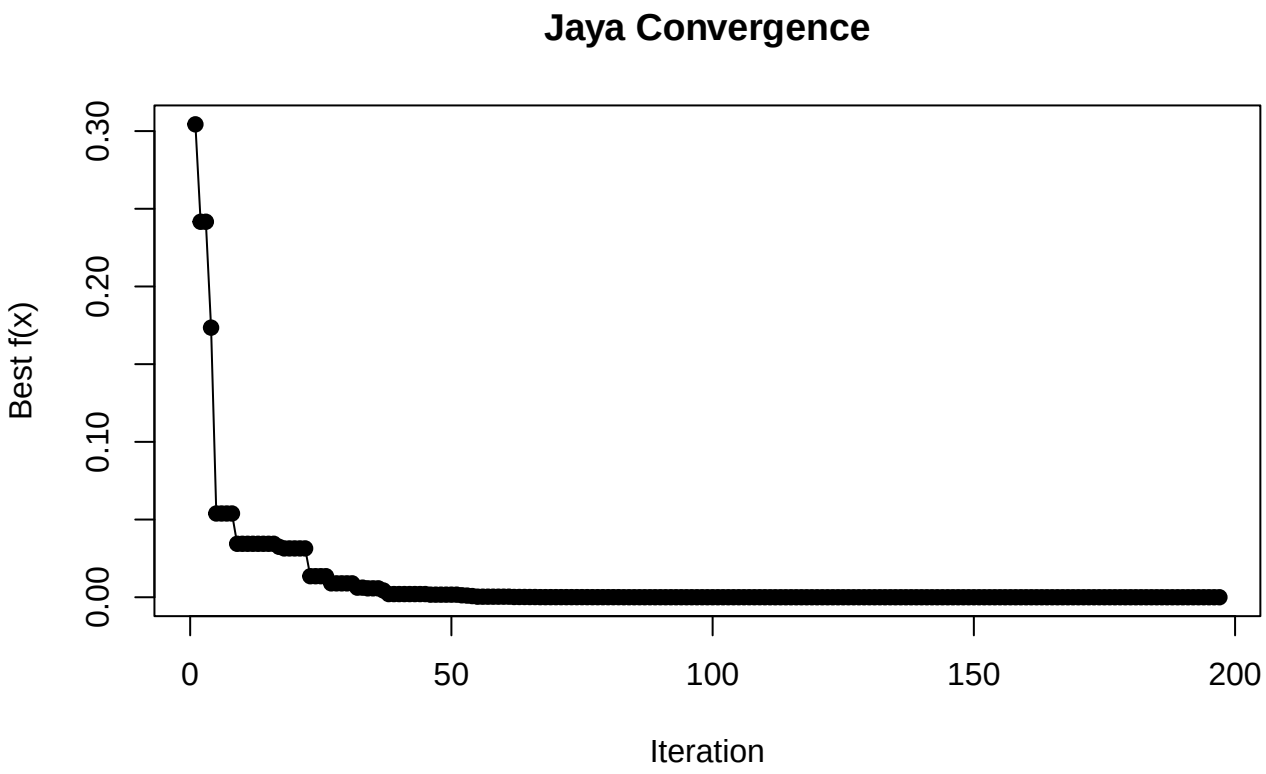

```
if (has_gg) print(autoplot.jaya(fit_reject))
```



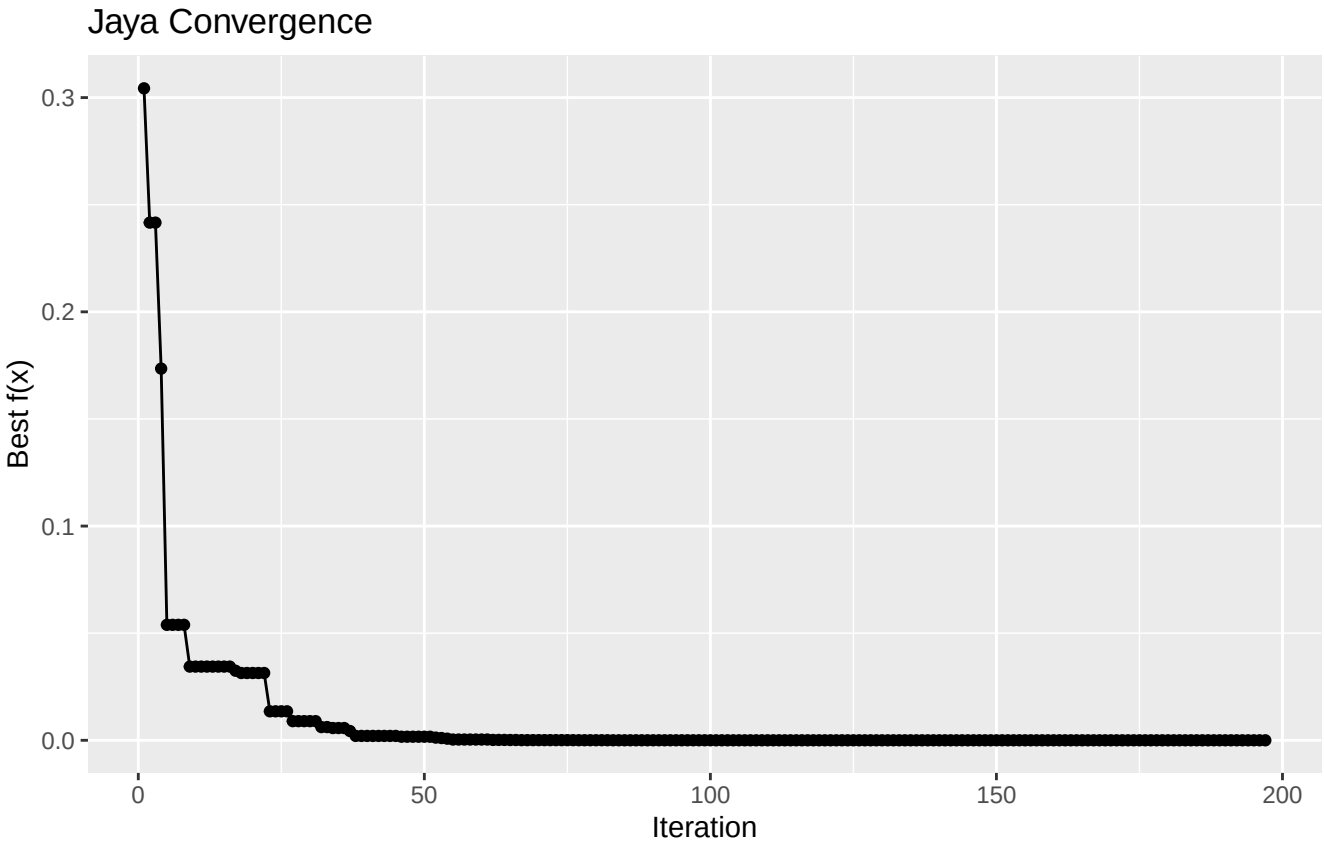

```r
# (B) Penalty for constraint violation
pen <- function(x) 100 * max(0, g1(x))  # linear penalty scaled up
set.seed(1)
fit_penalty <- jaya(fun = function(x) sphere(x) + pen(x),
                    lower = rep(-1, 5), upper = rep(1, 5),
                    popSize = 50, maxiter = 200)
summary(fit_penalty)
```

```
## Jaya (single-objective) summary
## Variables: 5 | popSize: 50 | maxiter: 200
## Best value: 2.474996087e-09
## Best parameters:
##            x1            x2            x3            x4            x5
##  1.377955e-06  6.631222e-06  2.163981e-06 -9.128758e-06 -4.887425e-06
## Evaluations: 9900 | Runtime: 0.030 s
```

```r
plot(fit_penalty)
```




```
if (has_gg) print(autoplot.jaya(fit_penalty))
```




---

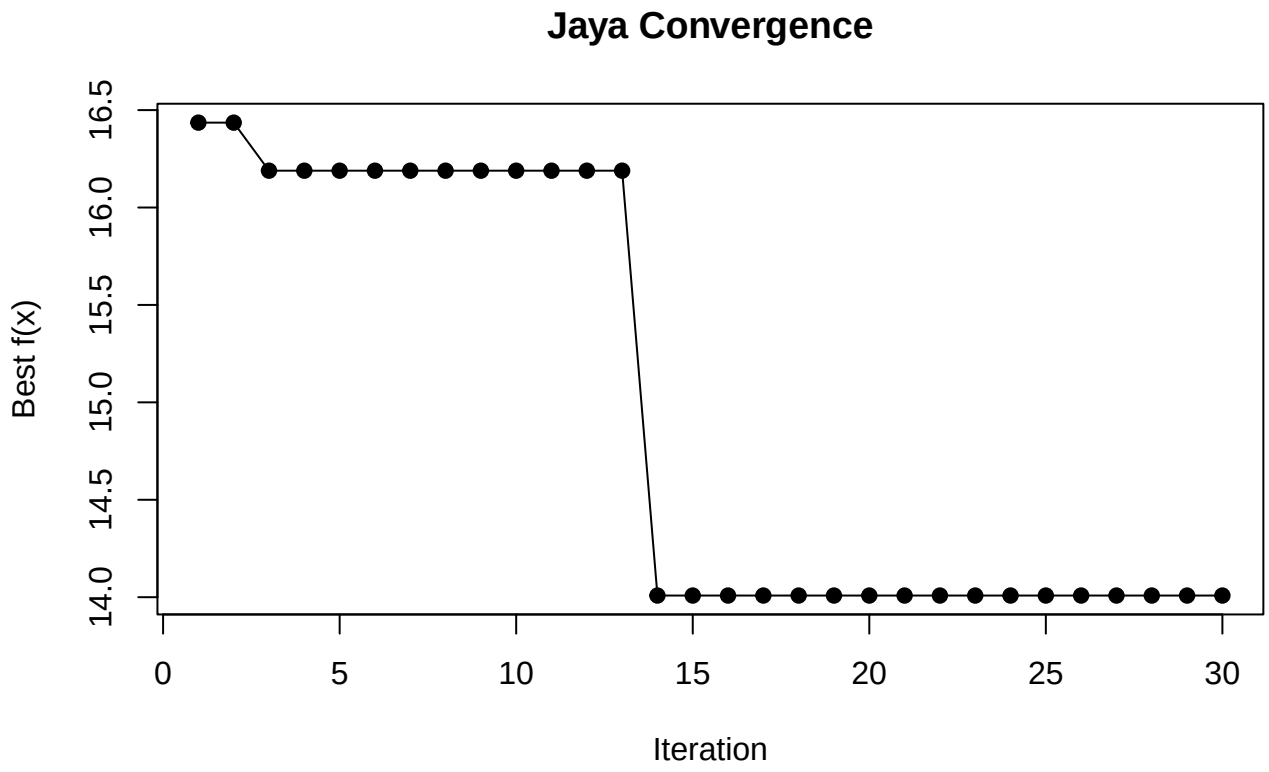
### 4. Single-Objective — Parallel Evaluation (Runtime Printed)

```
slow_ackley <- function(x) {
  Sys.sleep(0.002)  # emulate compute
  -20*exp(-0.2*sqrt(mean(x^2))) - exp(mean(cos(2*pi*x))) + 20 + exp(1)
}

fit_par <- jaya(fun = slow_ackley,
                lower = rep(-32.768, 8), upper = rep(32.768, 8),
                popSize = 100, maxiter = 30,
                parallel = TRUE, cores = max(1, parallel::detectCores()-1),
                verbose = TRUE)
summary(fit_par)
```

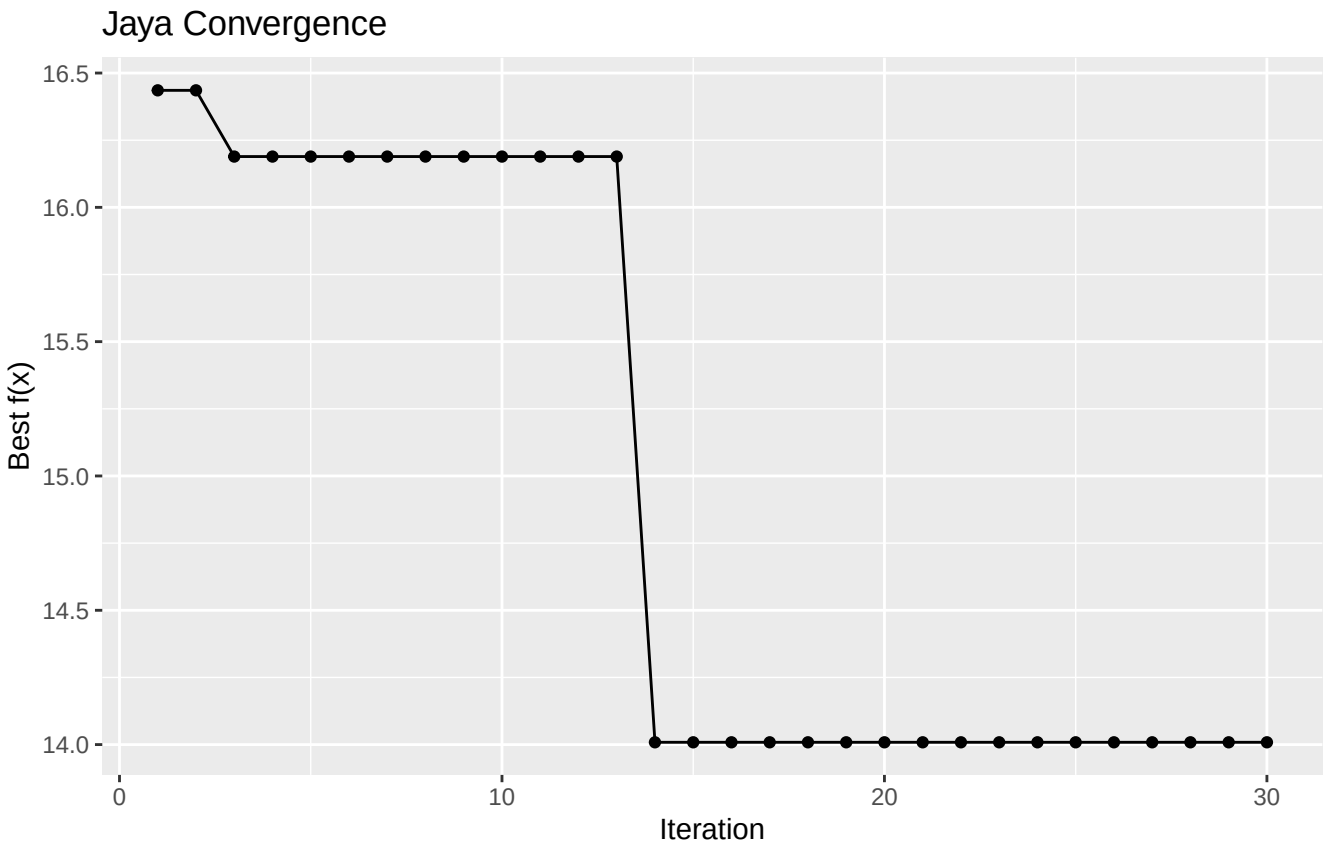
```
## Jaya (single-objective) summary
## Variables: 8 | popSize: 100 | maxiter: 30
## Best value: 14.00850491
## Best parameters:
##         x1         x2         x3         x4         x5         x6         x7
##  7.4795237 -4.8194673 -4.7518074  1.7598158 -0.3882949  3.9202823  2.5274273
##         x8
```

```
## -6.5598929
## Evaluations: 3100 | Runtime: 44.050 s
```

```
plot(fit_par)
```




```
if (has_gg) print(autoplot.jaya(fit_par))
```



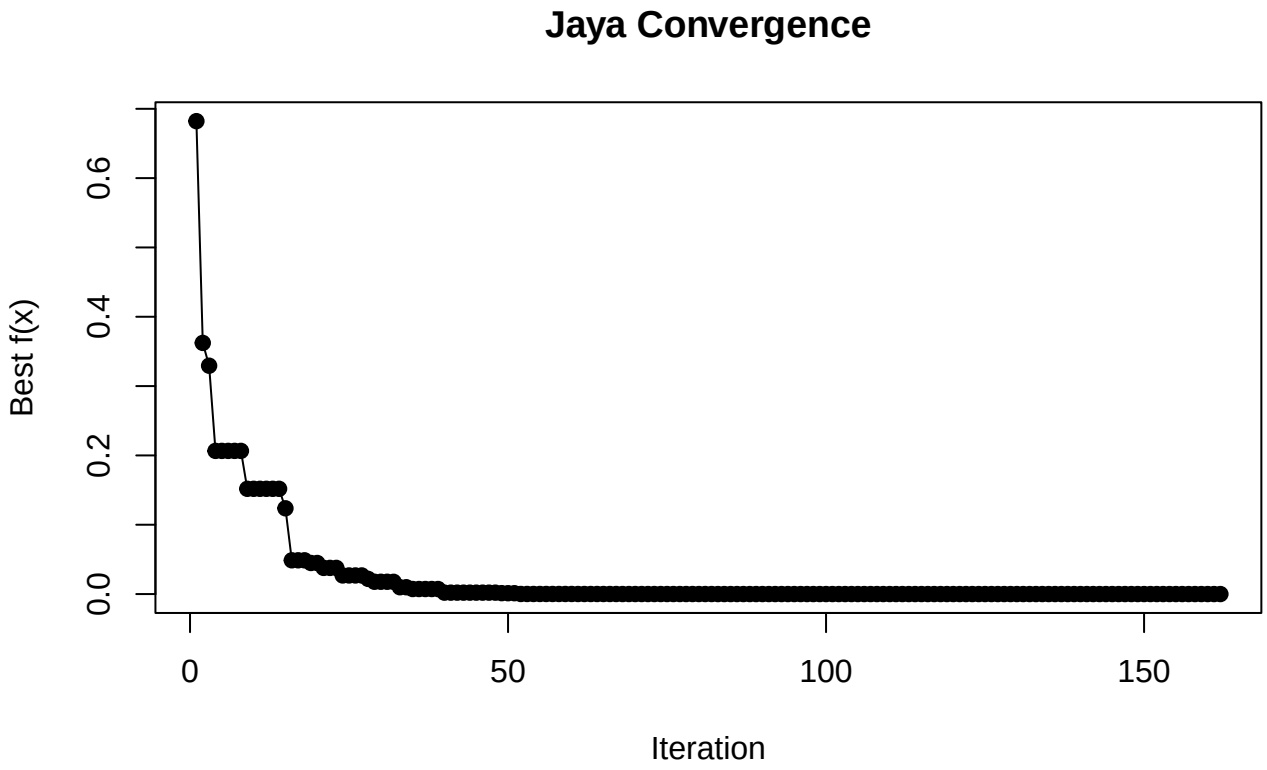

---

### 5. Single-Objective — Warm-Start with Suggestions (Results & Plot)

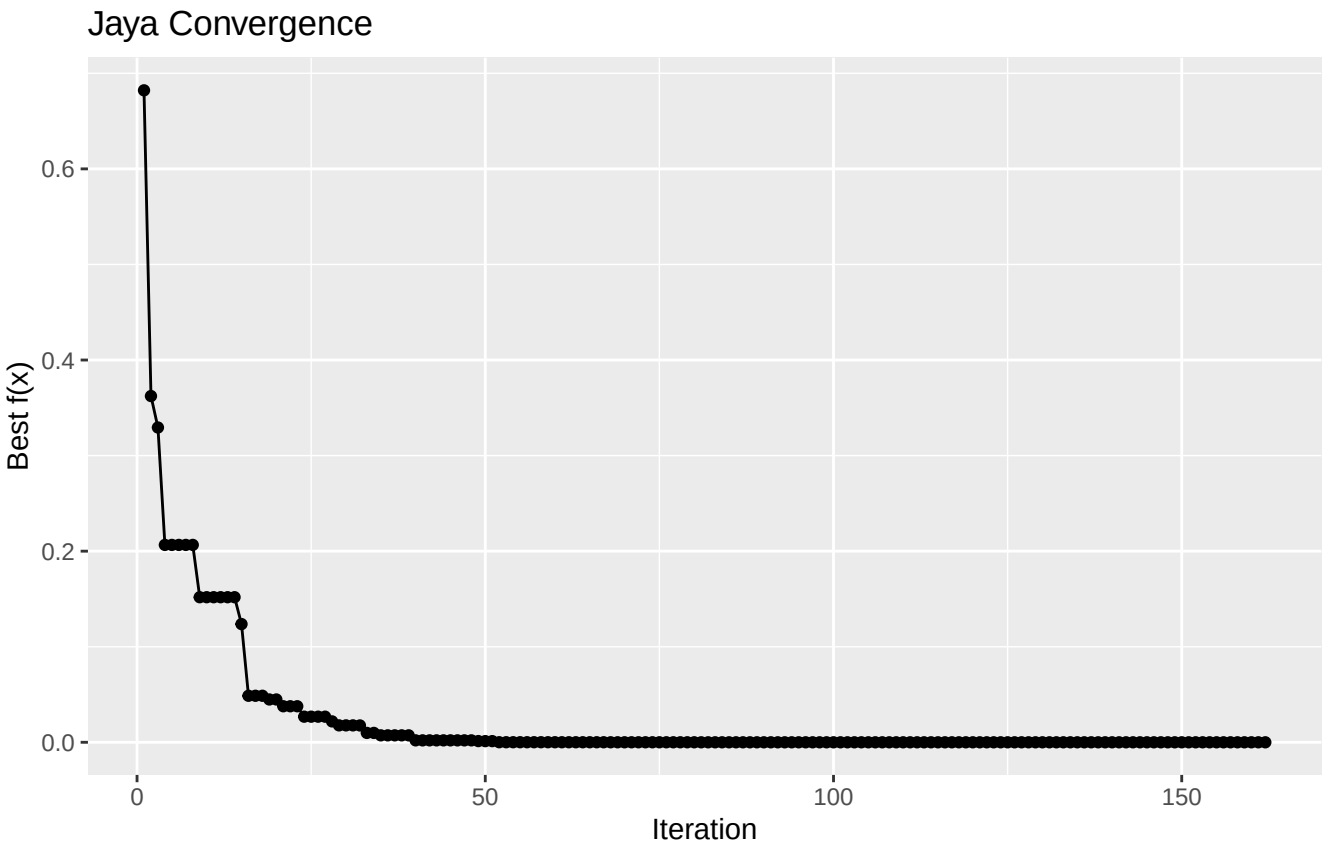
```
quad <- function(x) sum((x - 0.25)^2)
sug <- matrix(runif(20*5, -1, 1), ncol = 5)  # 20 suggested candidates

fit_ws <- jaya(fun = quad,
               lower = rep(-2, 5), upper = rep(2, 5),
               suggestions = sug,
               popSize = 60, maxiter = 200)
summary(fit_ws)
```

```
## Jaya (single-objective) summary
## Variables: 5 | popSize: 60 | maxiter: 200
## Best value: 2.763884713e-09
## Best parameters:
##        x1        x2        x3        x4        x5
## 0.2499996 0.2500042 0.2500087 0.2500039 0.2500041
## Evaluations: 9780 | Runtime: 0.030 s
```

```r
plot(fit_ws)
```




```r
if (has_gg) print(autoplot.jaya(fit_ws))
```



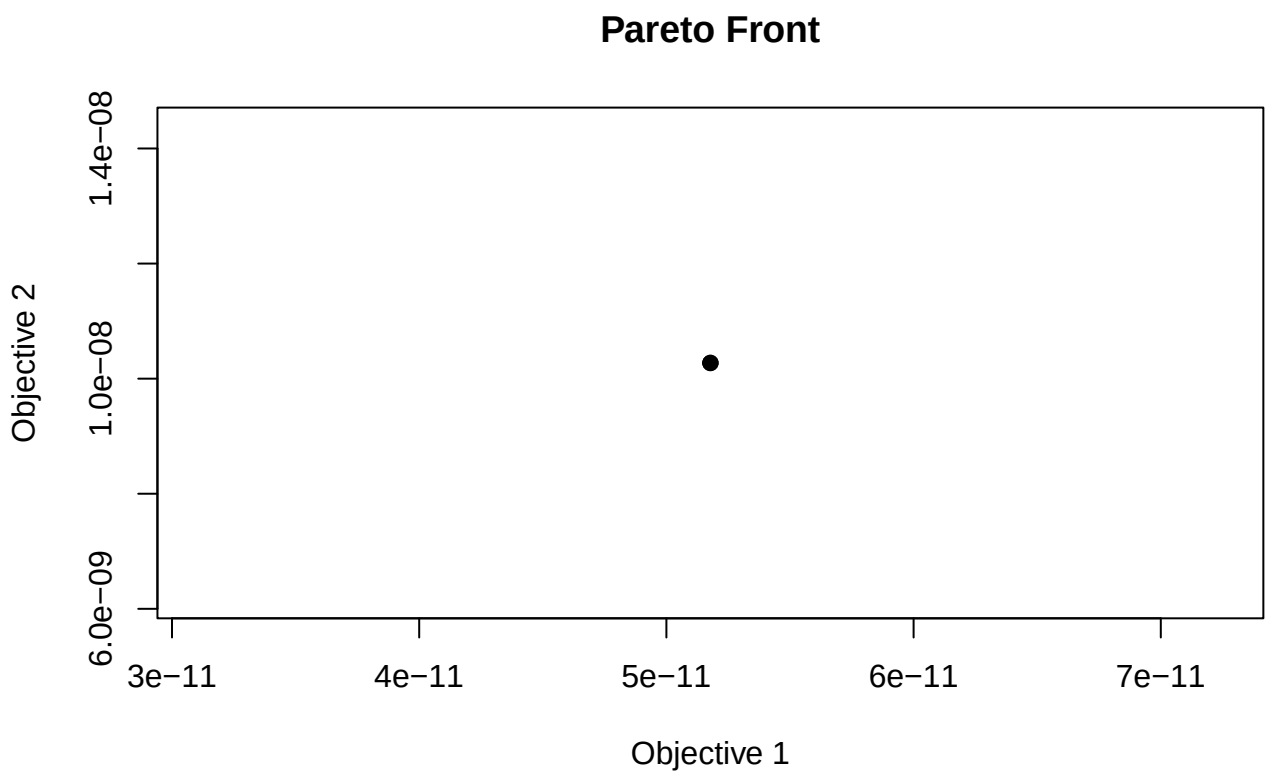

---

## 6. Multi-Objective (2 objectives) — Pareto Front, Plots & Tables

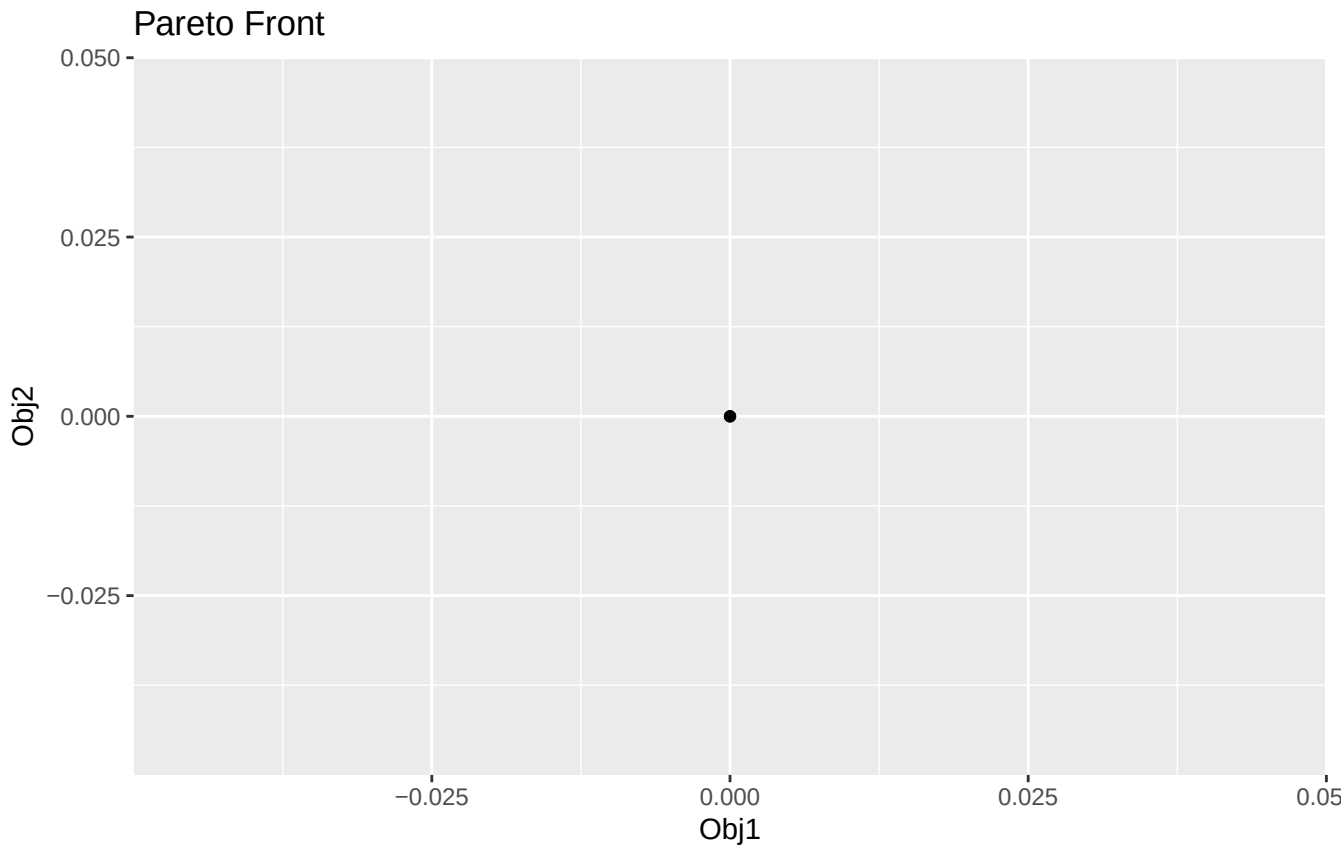
```
f1 <- function(x) sum(x^2)  # Sphere
f2 <- function(x) 10*length(x) + sum(x^2 - 10*cos(2*pi*x))  # Rastrigin

set.seed(7)
mo <- jaya_multi(objectives = list(f1, f2),
                 lower = rep(-5, 3), upper = rep(5, 3),
                 popSize = 120, maxiter = 200,
                 parallel = FALSE)
summary(mo)
```

```
## Jaya (multi-objective) summary
## Variables: 3 | popSize: 120 | maxiter: 200
## Pareto front (first 6 rows):
##          Obj1         Obj2
## 1 5.17801e-11 1.027276e-08
## 2 5.17801e-11 1.027276e-08
## Archive size: 2 | Evaluations: 24120 | Runtime: 8.260 s
```

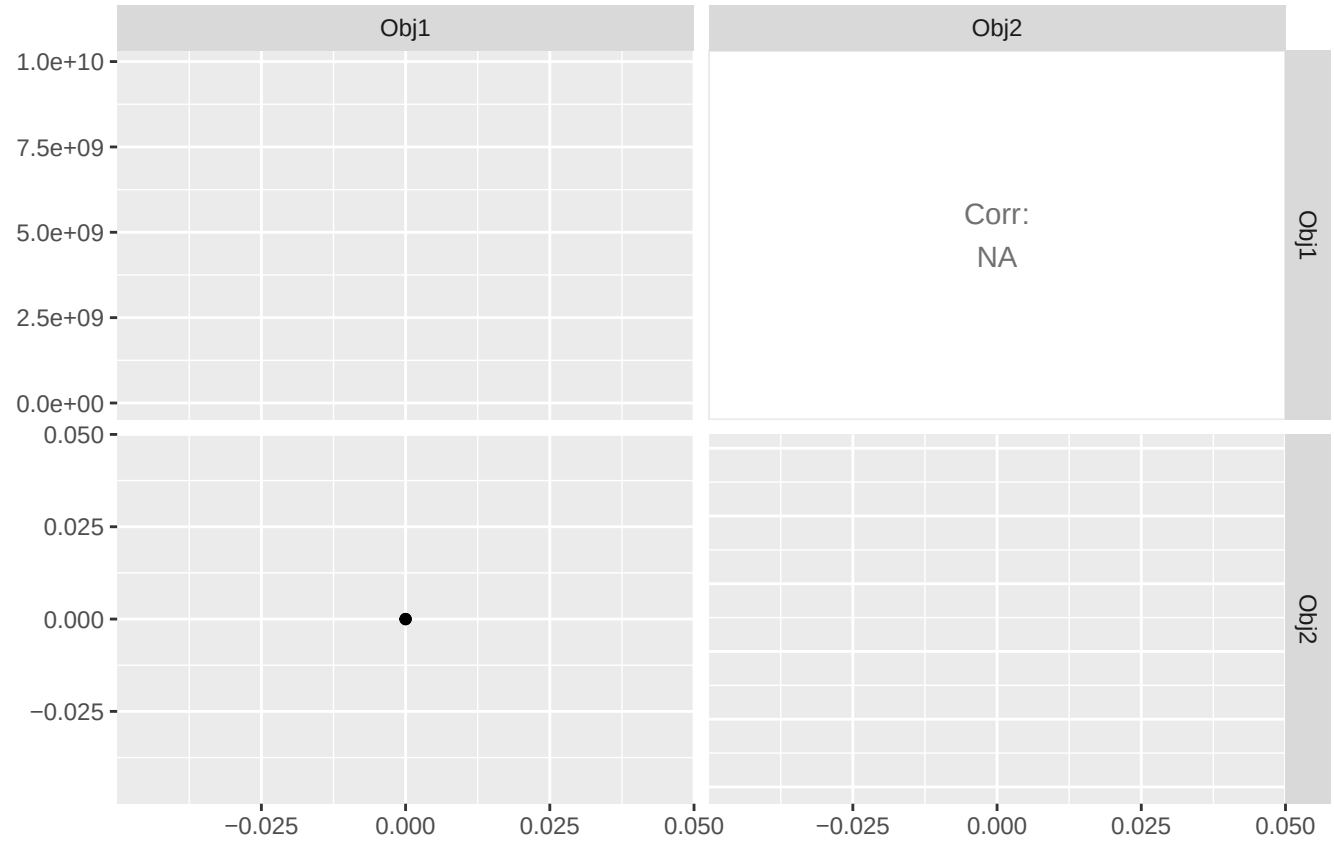
```r
# Base Pareto plot
plot(mo)
```




```r
# ggplot scatter
if (has_gg) print(autoplot.jaya_multi(mo, type = "scatter"))
```




```
# Pairs/parallel (if GGally is available); otherwise base pairs
if (has_ggally) {
  print(GGally::ggpairs(mo$Pareto_Front))
} else {
  pairs(mo$Pareto_Front, main = "Pareto Pairs (base)")
}
```




```r
# Show front / solutions head as a table
if (requireNamespace("knitr", quietly = TRUE)) {
  knitr::kable(head(mo$Pareto_Front), caption = "Pareto Front (head)")
  knitr::kable(head(mo$Solutions), caption = "Solutions (head)")
} else {
  head(mo$Pareto_Front); head(mo$Solutions)
}
```

Table 1: Solutions (head)

| x1 | x2 | x3 | Obj1 | Obj2 |
|---|---|---|---|---|
| 0 | 7.2e-06 | 0 | 0 | 0 |
| 0 | 7.2e-06 | 0 | 0 | 0 |

---

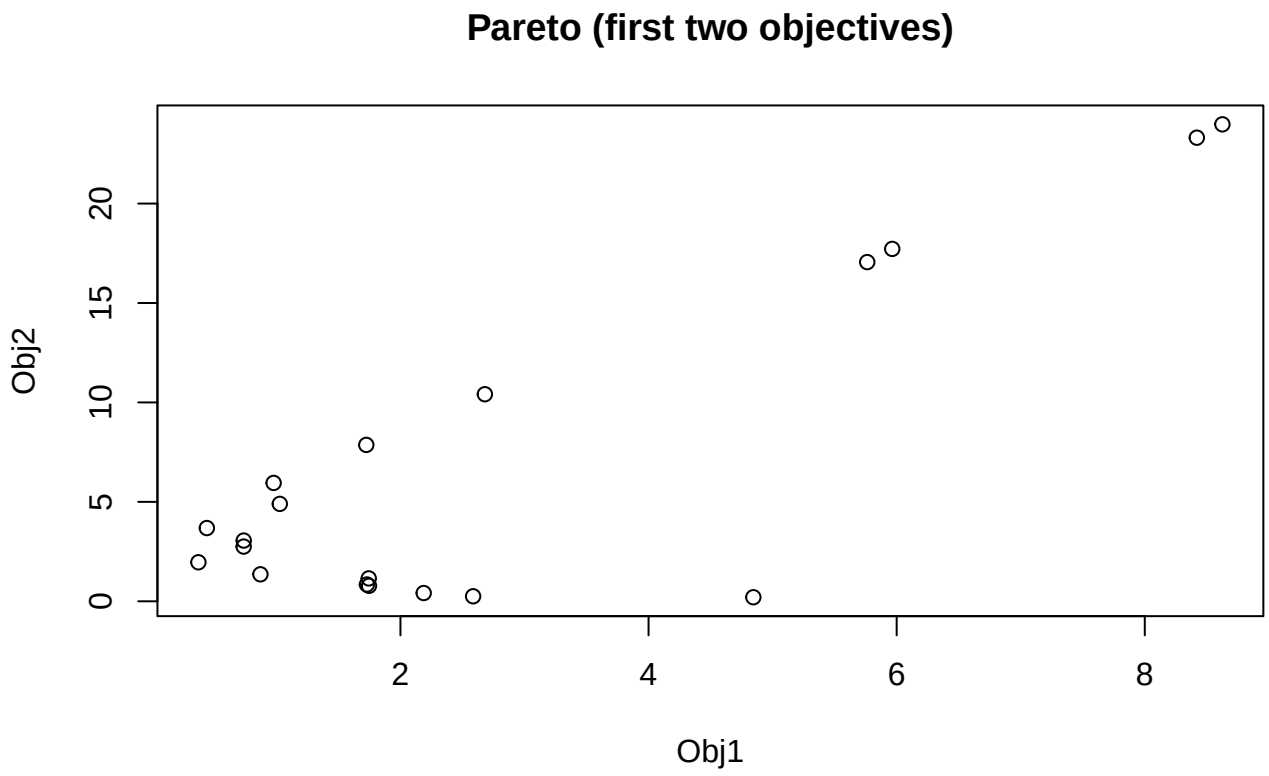
## 7. Multi-Objective (3 objectives) — -Dominance & Parallel (Archive Size Printed)

```r
f1 <- function(x) sum(x^2)
f2 <- function(x) sum((x - 1)^2)
f3 <- function(x) sum((x + 1)^2)
```

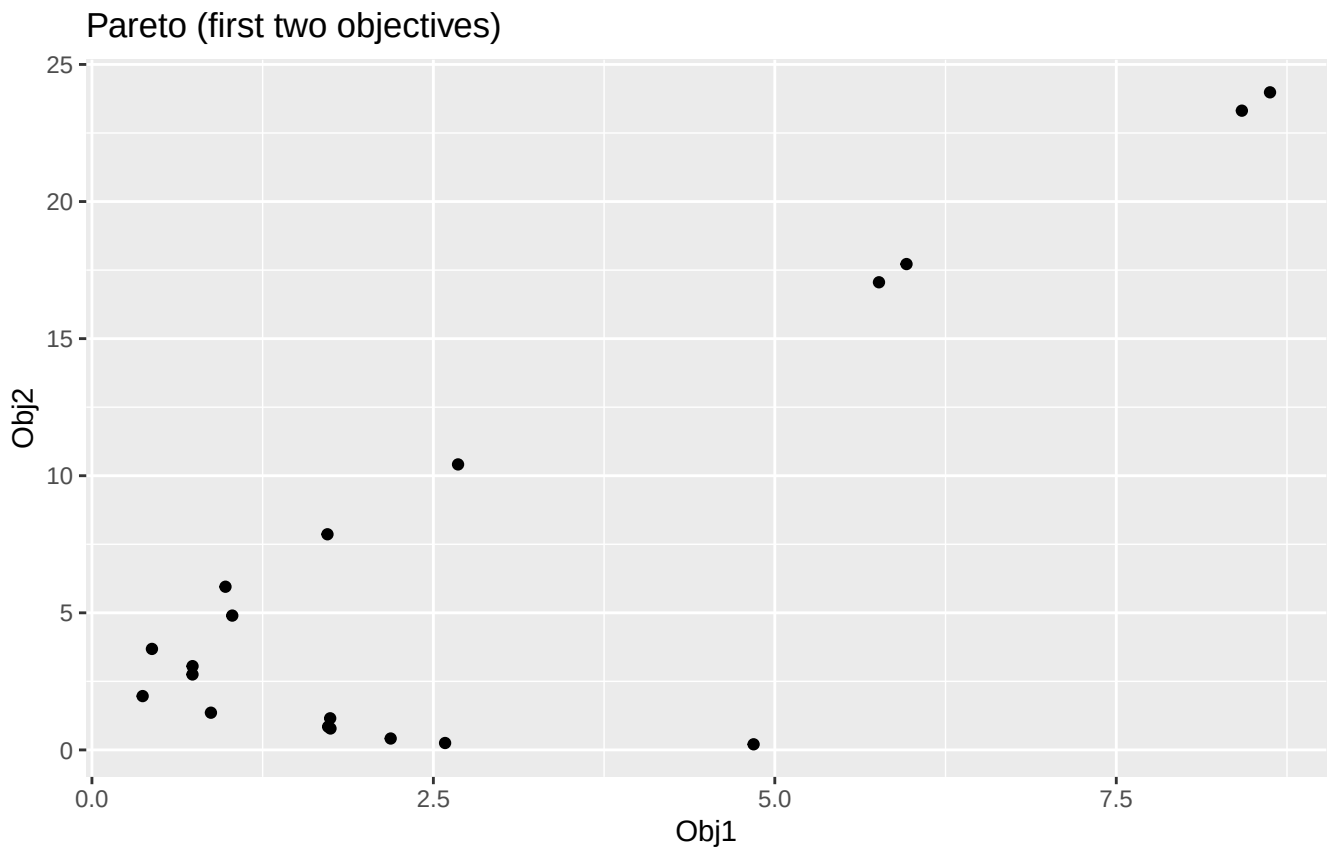
```r
set.seed(99)
mo3 <- jaya_multi(objectives = list(f1, f2, f3),
                  lower = rep(-3, 4), upper = rep(3, 4),
                  popSize = 160, maxiter = 20,
                  archive_size = 120,
                  epsilon = 0.05,      # coarsen grid, sparser set
                  parallel = TRUE, cores = max(1, parallel::detectCores()-1),
                  verbose = TRUE)
summary(mo3)
```

```
## Jaya (multi-objective) summary
## Variables: 4 | popSize: 160 | maxiter: 20
## Pareto front (first 6 rows):
##        Obj1      Obj2     Obj3
## 1 0.4394165  3.682927 5.195906
## 2 0.9777537  5.951717 4.003790
## 3 8.4196660 23.313753 1.525579
## 4 8.6257446 23.983133 1.268356
## 5 2.6806883 10.412878 2.948498
## 6 1.7246176  7.864115 3.585120
## Archive size: 19 | Evaluations: 3360 | Runtime: 4.280 s
```

```r
# Visual (plot first two objectives)
plot(data.frame(Obj1 = mo3$Pareto_Front[,1], Obj2 = mo3$Pareto_Front[,2]),
     xlab = "Obj1", ylab = "Obj2", main = "Pareto (first two objectives)")
```




```
if (has_gg) {
  print(ggplot2::ggplot(mo3$Pareto_Front, ggplot2::aes(Obj1, Obj2)) + ggplot2::geom_point() +
          ggplot2::labs(title = "Pareto (first two objectives)"))
}
```




## 8. Tidy/Broom-Style Tables (Printed)

```
# Single-objective tidy
rastrigin <- function(x) 10*length(x) + sum(x^2 - 10*cos(2*pi*x))
sfit <- jaya(rastrigin, rep(-5.12, 5), rep(5.12, 5), popSize = 60, maxiter = 150)

df_tidy <- tidy.jaya(sfit)      # per-iteration best
if (requireNamespace("knitr", quietly = TRUE)) knitr::kable(head(df_tidy), caption = "Per-iteration bes
```

Table 2: Per-iteration best (head)

| iter | best |
|---|---|
| 1 | 32.80589 |
| 2 | 32.80589 |
| 3 | 32.18290 |
| 4 | 32.18290 |
| 5 | 28.61049 |
| 6 | 22.02937 |

```r
gl_s <- glance.jaya(sfit)
if (requireNamespace("knitr", quietly = TRUE)) knitr::kable(gl_s, caption = "Glance (single-objective)")
```

Table 3: Glance (single-objective)

| best | iterations | n_eval | runtime_sec |
|---|---|---|---|
| 13.50183 | 56 | 3420 | 0.02 |

```r
# Multi-objective tidy
f1 <- function(x) sum(x^2); f2 <- function(x) sum((x-1)^2)
mfit <- jaya_multi(list(f1, f2), rep(-2, 3), rep(2, 3), popSize = 100, maxiter = 150)

pf  <- tidy.jaya_multi(mfit)   # Pareto front with solution_id
gl  <- glance.jaya_multi(mfit) # archive size, iterations, evals, runtime
if (requireNamespace("knitr", quietly = TRUE)) {
  knitr::kable(head(pf), caption = "Pareto front (head)")
  knitr::kable(gl, caption = "Glance (multi-objective)")
} else {
  head(pf); gl
}
```

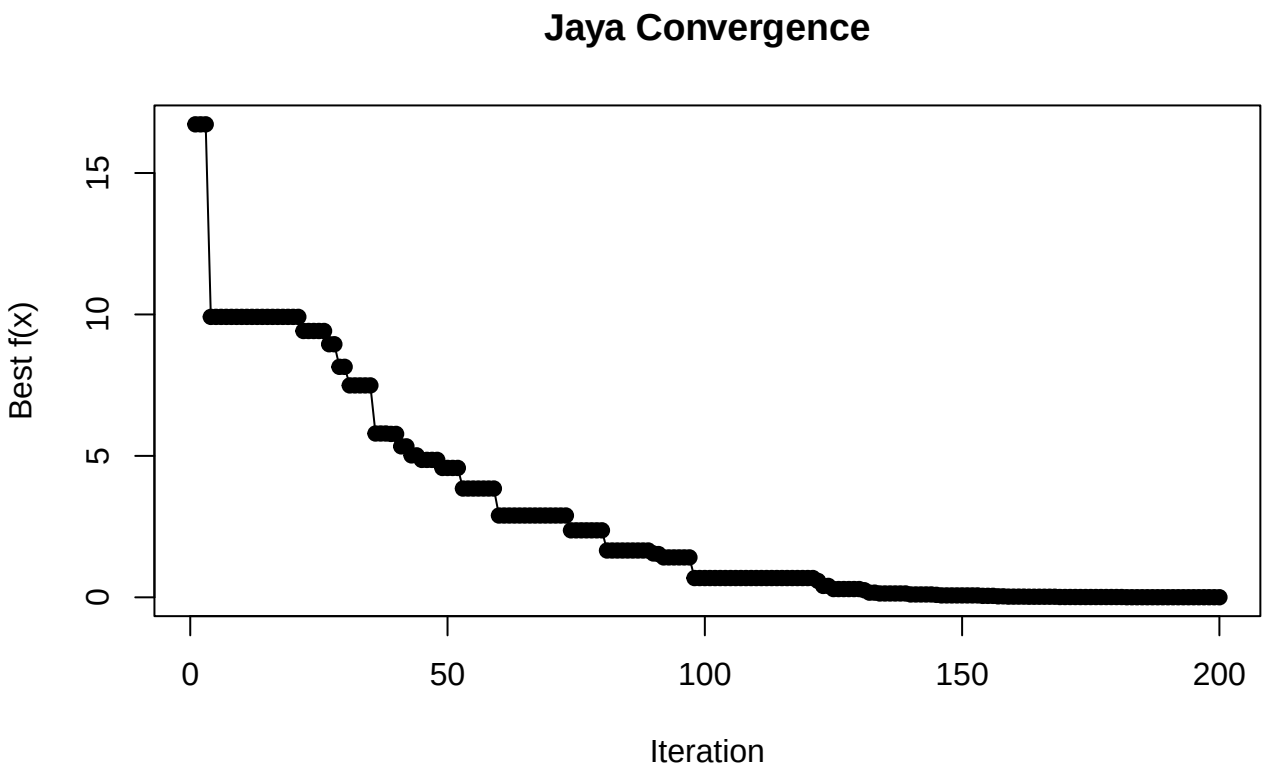
Table 4: Glance (multi-objective)

| archive | iterations | n_eval | runtime_sec |
|---|---|---|---|
| 100 | 150 | 15100 | 15.95 |

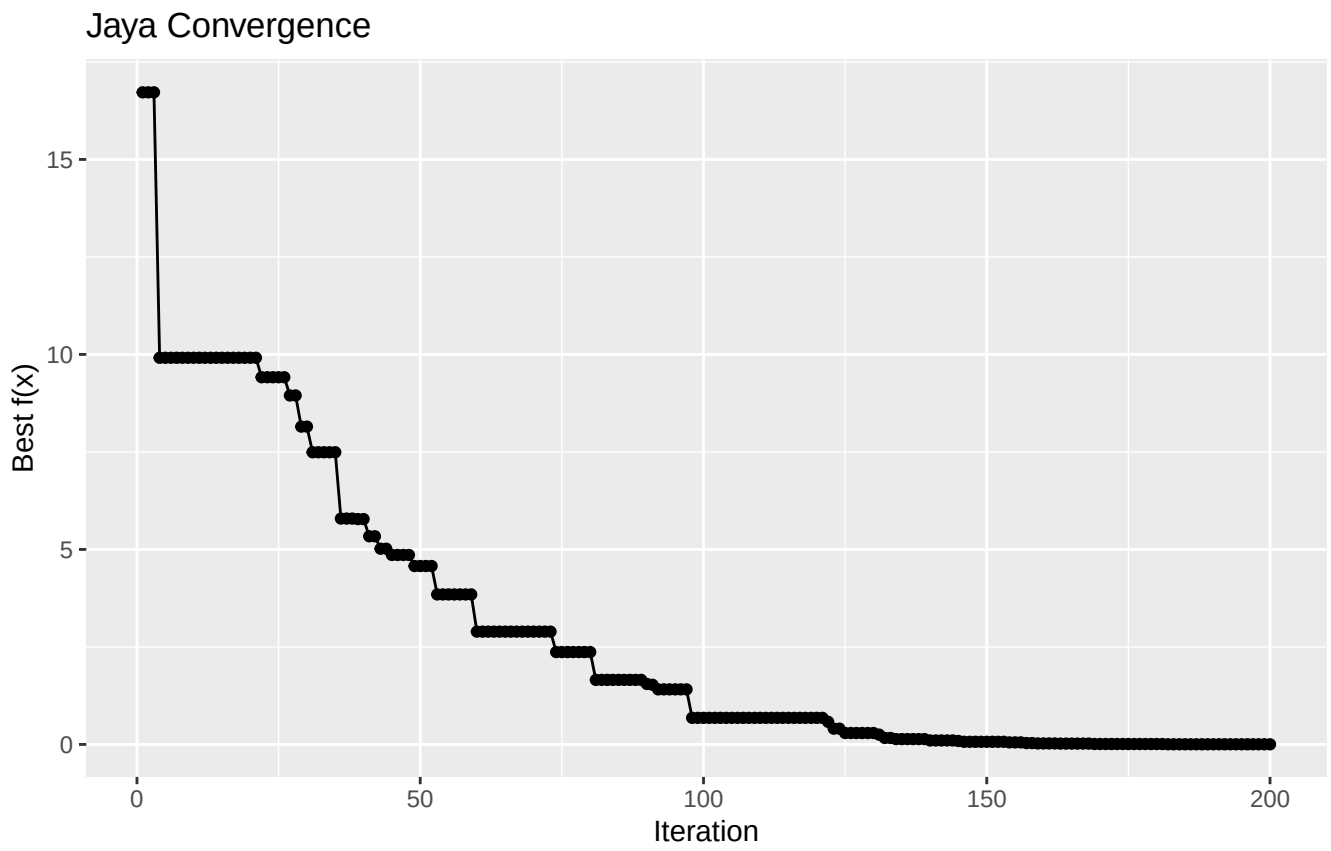
---

## 9. Saving Figures (PDF/PNG) and Showing Them Inline

```r
ackley <- function(x) -20*exp(-0.2*sqrt(mean(x^2))) - exp(mean(cos(2*pi*x))) + 20 + exp(1)
fit <- jaya(ackley, rep(-32.7, 5), rep(32.7, 5), popSize = 50, maxiter = 200)

# Show inline
plot(fit)
```




```
if (has_gg) print(autoplot.jaya(fit))
```




```
# Save to files for export
save_plot(function() plot(fit), file = "convergence.pdf", width = 6, height = 4)
if (has_gg) {
  p <- autoplot.jaya(fit)
  save_plot(p, file = "convergence.png", width = 6, height = 4)
}

# Confirm files exist (prints TRUE/FALSE)
file.exists("convergence.pdf")
```

```
## [1] TRUE
```

```
file.exists("convergence.png")
```

```
## [1] TRUE
```

---

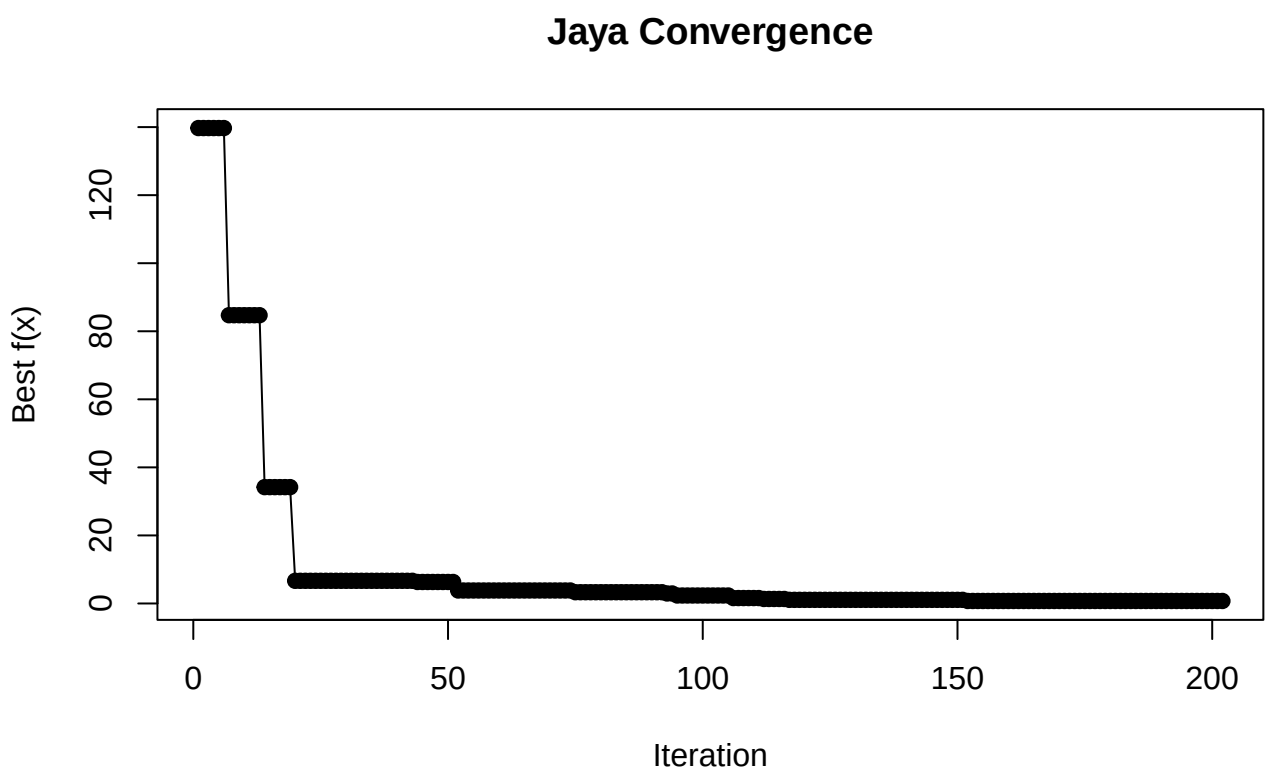
## 10. Early Stopping (Printed) + Plot

```
rosenbrock <- function(x) sum(100*(x[-1]-x[-length(x)]^2)^2 + (1 - x[-length(x)])^2)
```

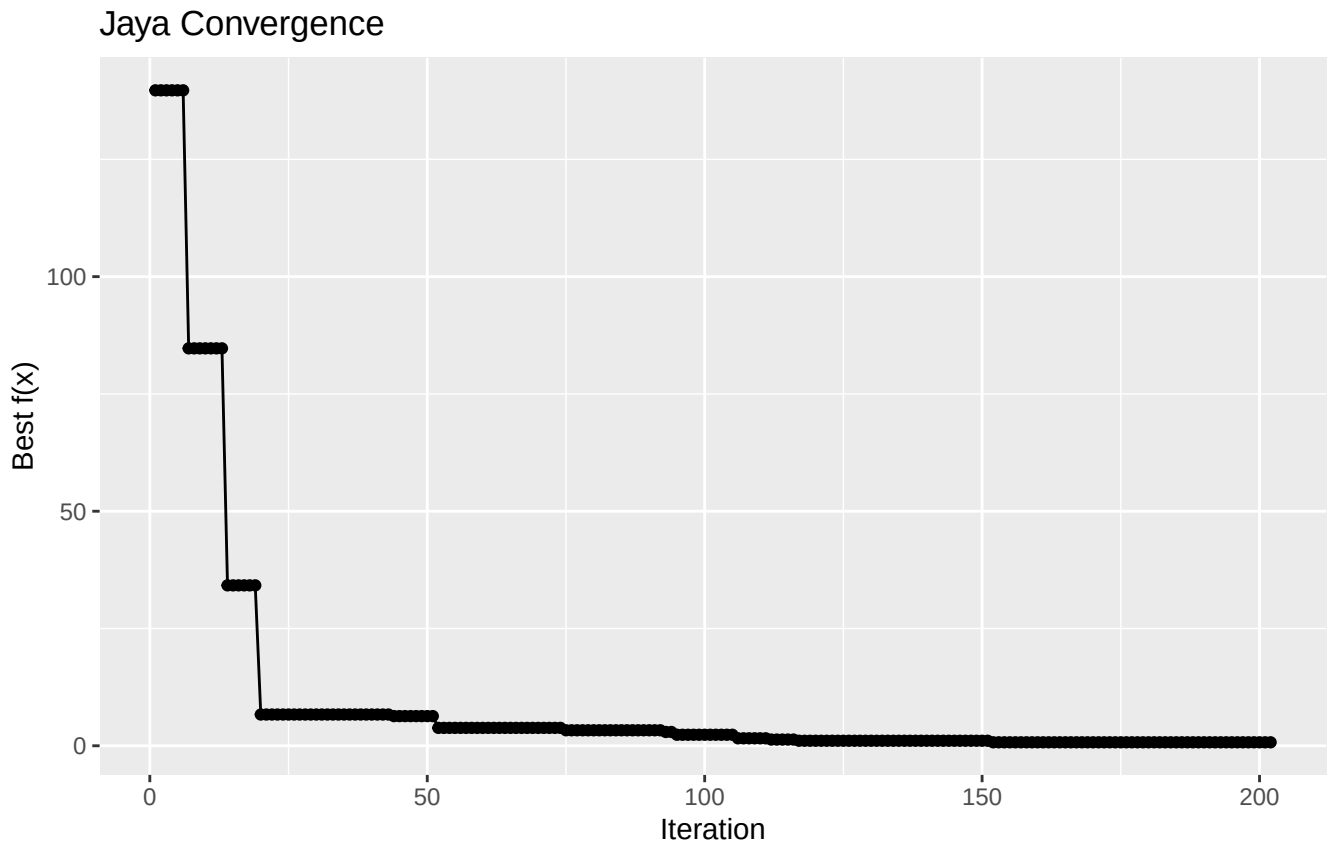
```r
fit_stop <- jaya(rosenbrock,
                 lower = rep(-2, 6), upper = rep(2, 6),
                 popSize = 80, maxiter = 1000,
                 seed = 123,
                 early_stopping = TRUE, tolerance = 1e-9, patience = 50,
                 verbose = TRUE)
summary(fit_stop)
```

```
## Jaya (single-objective) summary
## Variables: 6 | popSize: 80 | maxiter: 1000
## Best value: 0.7411294589
## Best parameters:
##        x1        x2        x3        x4        x5        x6
## 0.9388218 0.8829035 0.7761679 0.6290104 0.4334161 0.1897398
## Evaluations: 16240 | Runtime: 0.140 s
```

```r
plot(fit_stop)
```




```r
if (has_gg) print(autoplot.jaya(fit_stop))
```




---

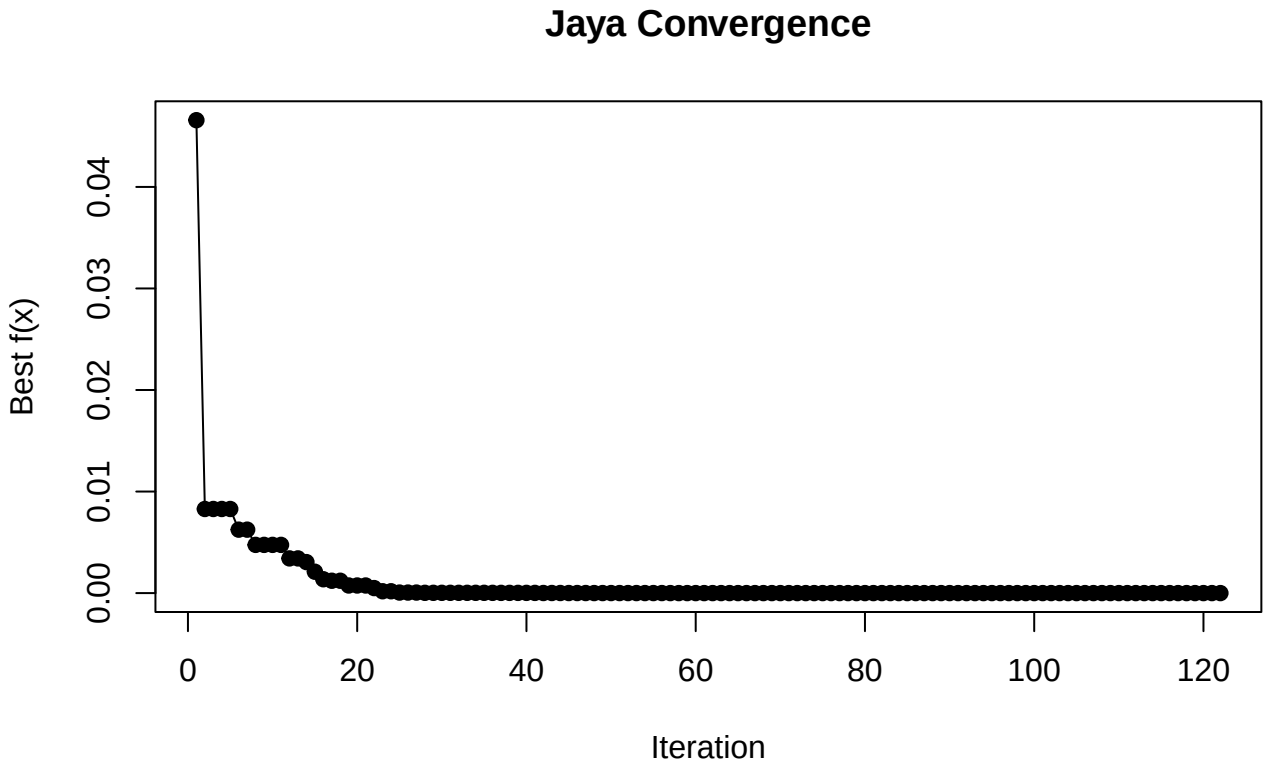
## 11. Constraint Strategies — Repair vs Reject (Results & Plots)

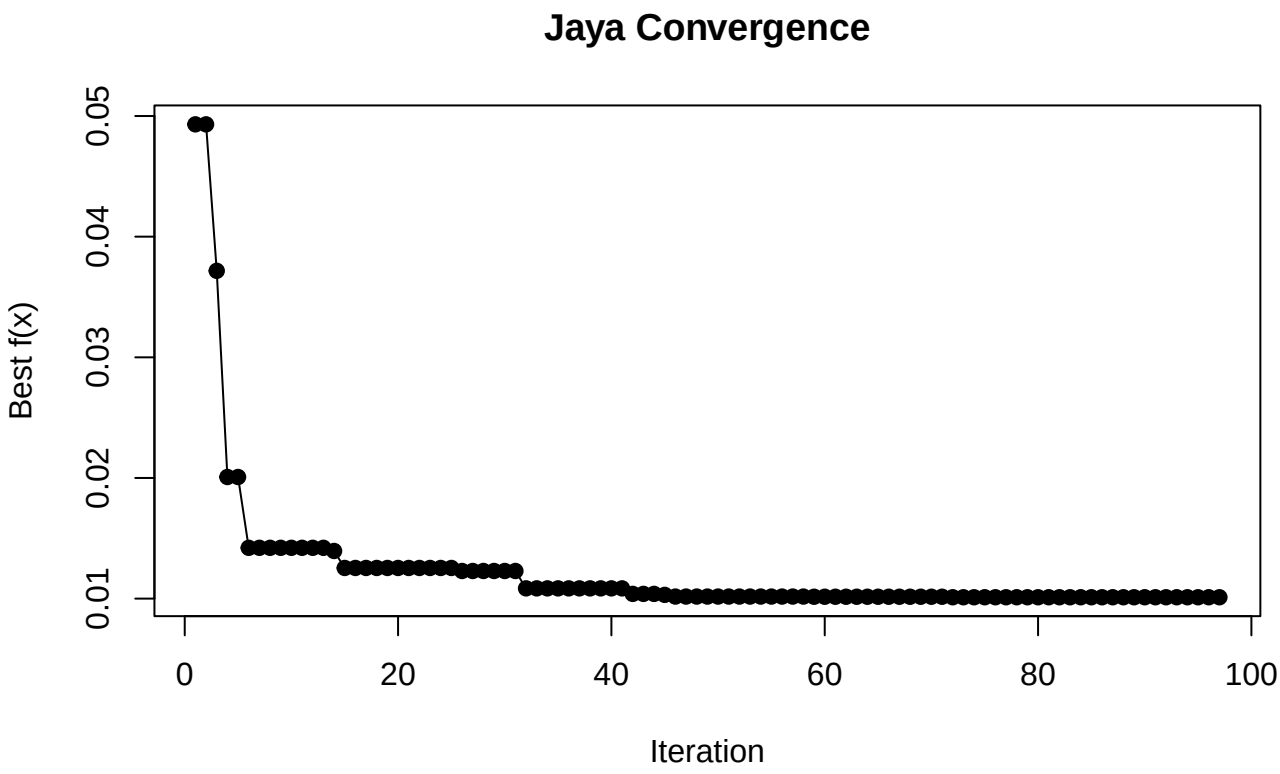
```
f <- function(x) sum((x - 0.3)^2)
g <- function(x)  sum(x) - 1  # <= 0

fit_repairish <- jaya(f, rep(0, 4), rep(1, 4),
                      constraints = list(g),
                      constraint_handling = "repair",
                      popSize = 50, maxiter = 200)

fit_reject2 <- jaya(f, rep(0, 4), rep(1, 4),
                    constraints = list(g),
                    constraint_handling = "reject",
                    popSize = 50, maxiter = 200)

summary(fit_repairish); summary(fit_reject2)
```

```
## Jaya (single-objective) summary
## Variables: 4 | popSize: 50 | maxiter: 200
## Best value: 7.534084953e-10
## Best parameters:
```

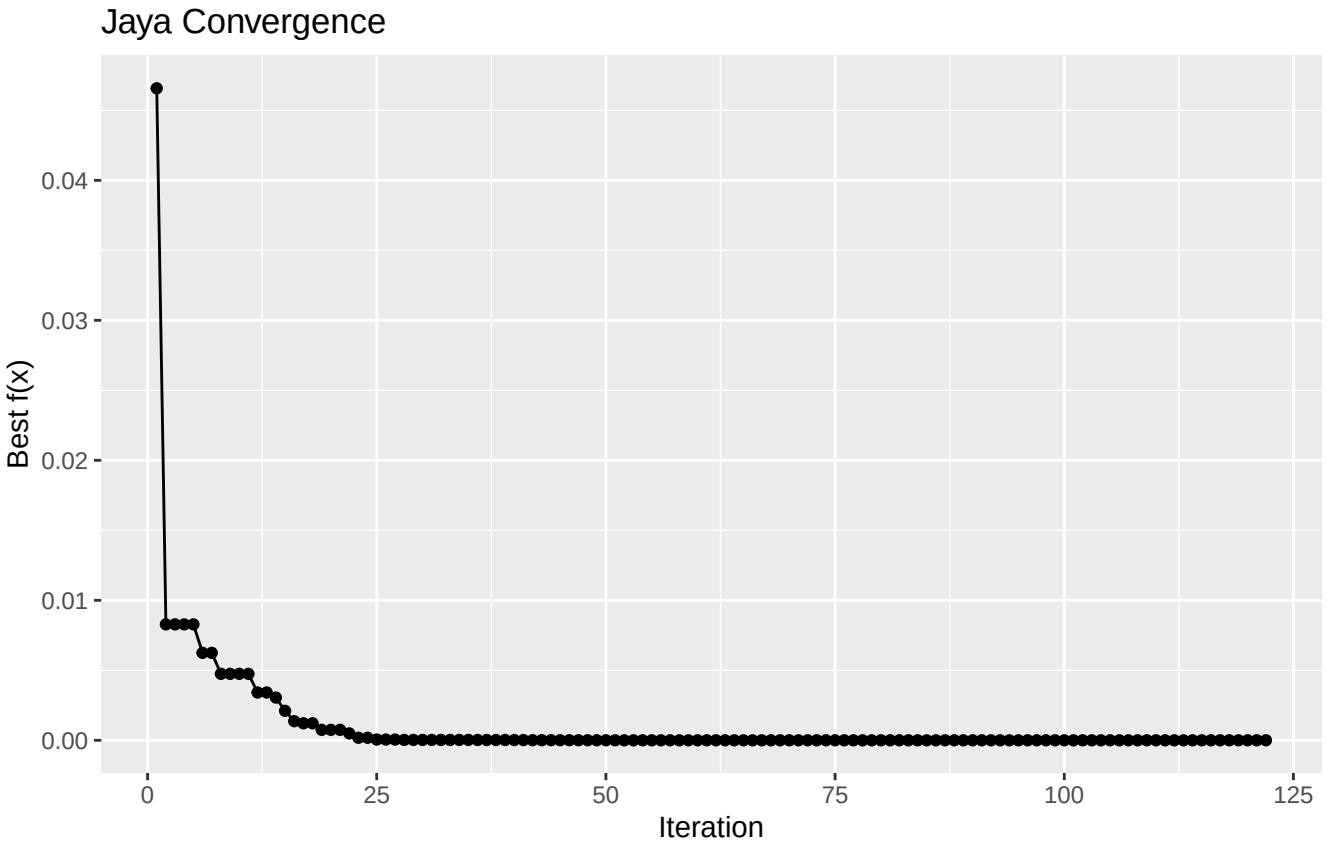
```
##        x1        x2        x3        x4
## 0.2999997 0.2999998 0.2999992 0.2999998
## Evaluations: 6150 | Runtime: 0.070 s

## Jaya (single-objective) summary
## Variables: 4 | popSize: 50 | maxiter: 200
## Best value: 0.01011381813
## Best parameters:
##        x1        x2        x3        x4
## 0.2567269 0.2500061 0.2475379 0.2453228
## Evaluations: 6251 | Runtime: 0.070 s
```

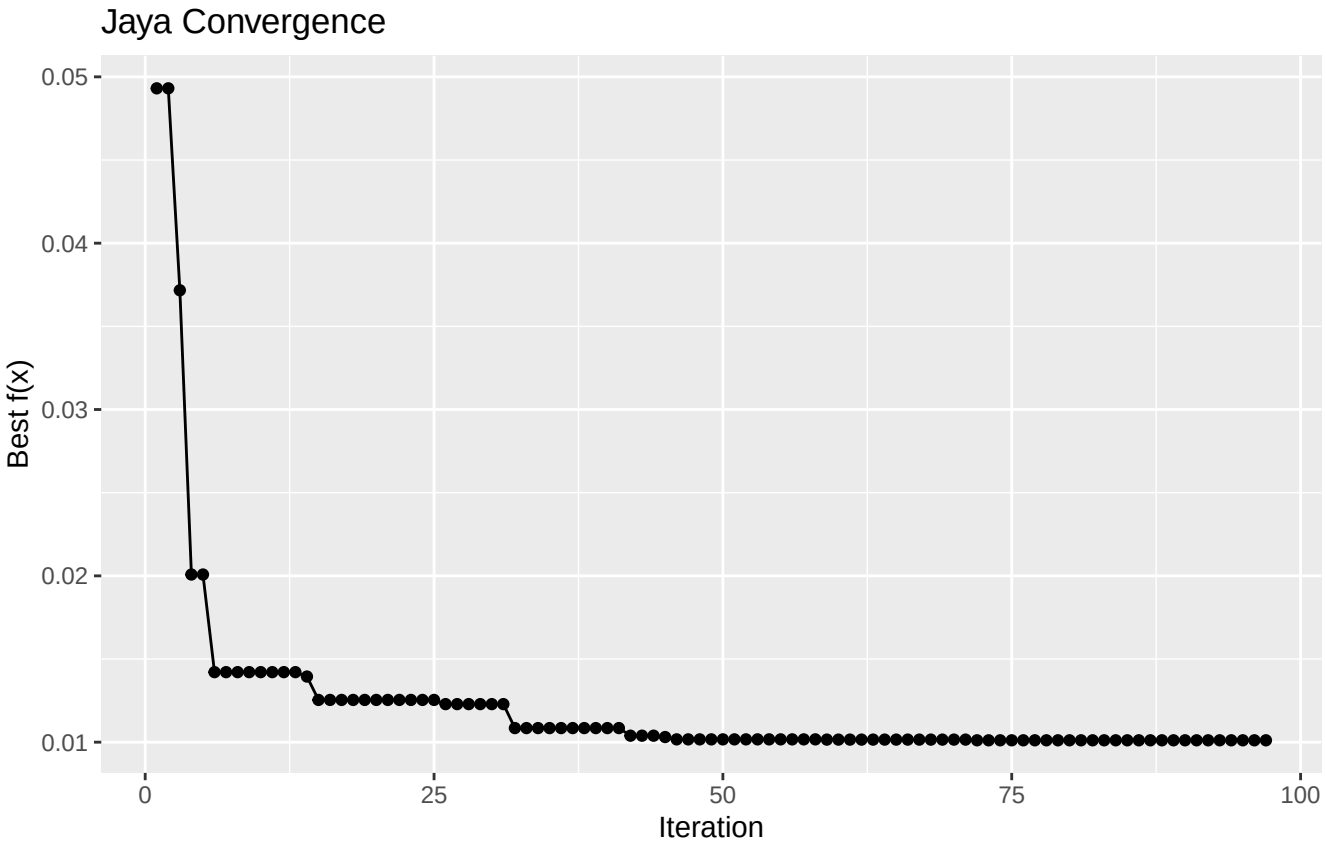
```
plot(fit_repairish); plot(fit_reject2)
```






```
if (has_gg) { print(autoplot.jaya(fit_repairish)); print(autoplot.jaya(fit_reject2)) }
```


Jaya Convergence
Best f(x)
0.04
0.03
0.02
0.01
0.00
0
25
50
75
100
125
Iteration




---

## 12. What's New in This Version (vs 1.0.3)

**Performance**

- **Vectorized core updates** for candidate generation and clamping, reducing loops and allocations.
- **Low-overhead parallelism**: single `future` plan per run with `future.apply` batching (Windows/macOS/Linux).
- **Early stopping**: halts when improvement stays below `tolerance` for `patience` iterations.

**Constraints**

- **`constraint_handling` modes**:
  - `"reject"` — enforce hard feasibility (resample at init; keep previous solution during search if proposal is infeasible).
  - `"repair"` — always repair to box bounds; ready for custom projection hooks.
  - `"penalty"` — allow exploration with soft feasibility via `penalty_fun(x)` (default linear penalty).

**Multi-Objective**

- **NSGA-II–style tools**: fast non-dominated sorting, crowding distance, and an elitist archive.
- **-dominance (optional)**: thins very large fronts while preserving spread.
- **Normalized guidance** (ideal/nadir) to pick global best/worst exemplars each iteration.

### API & UX

- **Aliases**: `jaya_min()` and `jaya_max()` for explicit intent.
- **Cleaner outputs**: `history`, `best_path`, `n_eval`, `runtime_sec`, and the original `call` for reproducibility.

### Plots & Tables

- **Visuals**: base plots plus `autoplot.jaya()` / `autoplot.jaya_multi()` when **ggplot2** is available.
- **Export**: `save_plot()` for PDF/PNG figures.
- **Reporting**: `summary()`, `tidy()`, `glance()` tables for quick inclusion in reports and pipelines.

---

## 13. Session Info

```
sessionInfo()
```

```
## R version 4.4.1 (2024-06-14 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_India.utf8  LC_CTYPE=English_India.utf8
## [3] LC_MONETARY=English_India.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_India.utf8
##
## time zone: Asia/Dubai
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] future_1.67.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.5        future.apply_1.20.0 dplyr_1.1.4
##  [4] compiler_4.4.1      tidyselect_1.2.1    tinytex_0.53
##  [7] ggstats_0.10.0      parallel_4.4.1      tidyr_1.3.1
## [10] globals_0.18.0      scales_1.3.0        yaml_2.3.8
## [13] fastmap_1.2.0       GGally_2.3.0        ggplot2_3.5.2
## [16] R6_2.5.1            labeling_0.4.3      generics_0.1.3
## [19] knitr_1.48          tibble_3.2.1        munsell_0.5.1
## [22] pillar_1.9.0        RColorBrewer_1.1-3  rlang_1.1.4
## [25] utf8_1.2.4          xfun_0.45           S7_0.2.0
## [28] cli_3.6.3           withr_3.0.1         magrittr_2.0.3
## [31] digest_0.6.36       grid_4.4.1          rstudioapi_0.16.0
```

```
## [34] lifecycle_1.0.4     vctrs_0.6.5         evaluate_1.0.1
## [37] glue_1.7.0          listenv_0.9.1       farver_2.1.2
## [40] codetools_0.2-20    parallelly_1.45.1   fansi_1.0.6
## [43] colorspace_2.1-0    rmarkdown_2.29      purrr_1.0.2
## [46] tools_4.4.1         pkgconfig_2.0.3     htmltools_0.5.8.1
```