Package 'swjm'

April 21, 2026

Title Stagewise Variable Selection for Joint Models of Semi-Competing Risks

Version 0.1.0

Description Implements stagewise regression for variable selection in joint models of recurrent events and terminal events (semi-competing risks). Supports two model frameworks: the joint frailty model (Cox-type) and the joint scale-change model (AFT-type). Provides cooperative lasso, lasso, and group lasso penalties with cross-validation for tuning parameter selection via cross-fitted estimating equations.

Depends R (>= 3.5.0)

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, Matrix, stats, reReg

Suggests testthat (>= 3.0.0), survival, timeROC, knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Jared D. Huling [aut, cre],
Lingfeng Huo [aut],
Ziren Jiang [aut],
Jue Hou [aut],
Sy Han (Steven) Chiou [ctb],
Chiung-Yu Huang [ctb]

Maintainer Jared D. Huling <jaredhuling@gmail.com>

Repository CRAN

Date/Publication 2026-04-21 18:50:09 UTC

Contents

`swjm-package` *swjm: Stagewise Variable Selection for Joint Models of Semi-Competing Risks*

Description

Implements stagewise regression for penalized variable selection in joint models of recurrent events and terminal events (semi-competing risks).

Two model frameworks are supported:

- **Joint Frailty Model (JFM)**: Cox-type model where `alpha` = recurrence/readmission coefficients (first p elements of theta), `beta` = terminal/death coefficients (second p elements).
- **Joint Scale-Change Model (JSCM)**: AFT-type model where `alpha` = recurrence/readmission coefficients (first p elements of theta), `beta` = terminal/death coefficients (second p elements).

Three penalty types are available:

- `"coop"`: Cooperative lasso, which encourages shared support between the recurrence and terminal event coefficient vectors.
- `"lasso"`: Standard L1 penalty applied coordinate-wise.
- `"group"`: Group lasso, which selects variables jointly across both processes.

Data Format

All functions expect a recurrent-event data frame with the following columns:

id Subject identifier (integer).

t.start Interval start time.

t.stop Interval stop time.

event 1 for recurrent event, 0 for terminal/censoring row.

status 1 for death, 0 for alive/censored.

x1, x2, ..., xp Covariate columns.

Author(s)

Maintainer: Jared D. Huling <jaredhuling@gmail.com>

Authors:

- Lingfeng Huo
- Ziren Jiang
- Jue Hou

Other contributors:

- Sy Han (Steven) Chiou [contributor]
- Chiung-Yu Huang [contributor]

`baseline_hazard` *Cumulative Baseline Hazard Step Functions*

Description

Evaluates the cumulative baseline hazards for readmission and death at arbitrary time points. For JFM, Breslow-type estimators are used. For JSCM, Nelson-Aalen estimators on the accelerated time scale are used.

Usage

```
baseline_hazard(
  object,
  times = NULL,
  which = c("both", "readmission", "death")
)
```

Arguments

`object`	An object of class `"swjm_cv"`.
`times`	Numeric vector of evaluation times. If `NULL`, the observed event times stored in the fit are used.
`which`	Character. Which baseline hazard(s) to return: `"both"` (default), `"readmission"`, or `"death"`.

Value

A data frame with column `time` and, depending on `which`, `cumhaz_readmission` and/or `cumhaz_death`.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
cv  <- cv_stagewise(dat$data, model = "jfm", penalty = "coop",
                    max_iter = 100)
bh  <- baseline_hazard(cv)
head(bh)
```

`cv_stagewise` *Cross-Validation for Stagewise Variable Selection*

Description

Selects the optimal penalty parameter (lambda) along the stagewise path using K-fold cross-validation with cross-fitted estimating equations.

Usage

```
cv_stagewise(
  data,
  model = c("jfm", "jscm"),
  penalty = c("coop", "lasso", "group"),
  K = 5L,
  lambda_seq = NULL,
  eps = NULL,
  max_iter = NULL,
  pp = NULL,
  estimate_frailty = FALSE,
  ncores = 1L,
  standardize = TRUE
)
```

Arguments

`data`	A data frame in recurrent-event format.
`model`	Character. Either `"jfm"` or `"jscm"`.
`penalty`	Character. One of `"coop"`, `"lasso"`, or `"group"`.
`K`	Integer. Number of cross-validation folds (default 5).
`lambda_seq`	Numeric vector. The lambda sequence at which to evaluate the cross-validation criterion. If `NULL`, extracted from a full-data stagewise fit.
`eps`	Numeric. Step size (passed to `stagewise_fit`).
`max_iter`	Integer. Maximum iterations (passed to `stagewise_fit`).
`pp`	Integer. Early-stop block size (passed to `stagewise_fit`).

`estimate_frailty`
: Logical. For JFM only: if `TRUE`, estimates the frailty variance and uses frailty weights in the estimating equations (passed to `stagewise_fit`).

`ncores`
: Integer. Number of cores for parallel fold training (default 1, sequential). Uses `parallel::parLapply` with a PSOCK cluster, which works on all platforms including Windows.

`standardize`
: Logical. If `TRUE` (default), covariates are standardized before fitting (passed to `stagewise_fit`).

Value

An object of class `"swjm_cv"`, a list with components:

position_CF Integer, position of best lambda by combined cross-fitted score norm.

position_CF_re Integer, position of best lambda by recurrence score norm.

position_CF_cen Integer, position of best lambda by terminal score norm.

lambda_seq Numeric vector of lambda values evaluated.

Scorenorm_crossfit Combined cross-fitted score norm path.

Scorenorm_crossfit_re Recurrence score norm path.

Scorenorm_crossfit_ce Terminal score norm path.

full_fit The full-data stagewise fit (class `"swjm_path"`).

model Character.

penalty Character.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
fit <- stagewise_fit(dat$data, model = "jfm", penalty = "coop",
                     max_iter = 100)
cv_res <- cv_stagewise(dat$data, model = "jfm", penalty = "coop",
                       lambda_seq = fit$lambda, max_iter = 100)
cv_res
```

---

generate_data *Generate Simulated Data for Joint Models*

Description

Unified interface that dispatches to model-specific data generation functions for the joint frailty model (JFM) or joint scale-change model (JSCM).

Usage

```
generate_data(n, p, scenario = 1, model = c("jfm", "jscm"), ...)
```

Arguments

n	Integer. Number of subjects.
p	Integer. Number of covariates (should be a multiple of 10 for scenarios 1–3).
scenario	Integer. Scenario for true coefficient configuration (1, 2, 3, or other for a simple default).
model	Character. Either "jfm" for the joint frailty model or "jscm" for the joint scale-change model.
...	Additional arguments passed to the model-specific function. For JFM: b, lambda0_d, lambda0_r, gamma_frailty, cov_type. For JSCM: b.

Value

A list with components:

data A data frame in recurrent-event format with columns id, t.start, t.stop, event, status, and covariate columns x1, ..., xp.

alpha_true Numeric vector of true alpha coefficients.

beta_true Numeric vector of true beta coefficients.

Examples

```
# JFM data with 30 subjects and 10 covariates
dat_jfm <- generate_data(n = 30, p = 10, scenario = 1, model = "jfm")
head(dat_jfm$data)

# JSCM data with 30 subjects and 10 covariates
dat_jscm <- generate_data(n = 30, p = 10, scenario = 1, model = "jscm")
head(dat_jscm$data)
```

---

generate_data_jfm *Generate Simulated Data for the Joint Frailty Model (JFM)*

Description

Generates recurrent-event and terminal-event data under a Cox-type joint frailty model. Ported from Data_Generation_time_dependent_new().

Usage

```
generate_data_jfm(
  n,
  p,
  scenario = 1,
  b = 6.5,
  lambda0_d = 0.041,
```

```
  lambda0_r = 1,
  gamma_frailty = 0,
  cov_type = c("internal", "external", "fixed")
)
```

Arguments

n	Integer. Number of subjects.
p	Integer. Number of covariates.
scenario	Integer. Scenario (1, 2, 3, or other).
b	Numeric. Upper bound of the censoring uniform distribution (default 6.50).
lambda0_d	Numeric. Baseline hazard rate for the terminal event (default 0.041).
lambda0_r	Numeric. Baseline hazard rate for recurrent events (default 1).
gamma_frailty	Numeric. Frailty variance parameter. When positive, a subject-specific frailty $Z_i \sim \text{Gamma}(1/\gamma, 1/\gamma)$ is drawn for each subject and multiplies both hazard rates. When 0 (default), no frailty is used ($Z_i = 1$).
cov_type	Character. How time-varying covariates are generated: "internal" (default) redraws covariates at each recurrent event; "external" changes covariates at predetermined Poisson times independent of the event process (Kalbfleisch-compatible); "fixed" draws one covariate vector per subject that never changes.

Details

Internally the simulation uses the Rondeau et al. (2007) convention where alpha governs death and beta governs recurrence. The returned alpha_true and beta_true are relabelled to match the package-wide convention:

- alpha_true: recurrence (readmission) coefficients.
- beta_true: terminal (death) coefficients.

Within each subject the covariates are regenerated at each gap time, yielding time-dependent covariates. Censoring times are Uniform(1, b).

Value

A list with components:

data Data frame with columns id, t.start, t.stop, event, status, x1, ..., xp.

alpha_true True alpha (terminal) coefficients.

beta_true True beta (recurrence) coefficients.

Examples

```
dat <- generate_data_jfm(n = 30, p = 10, scenario = 1)
head(dat$data)
dat$alpha_true
dat$beta_true
```

`generate_data_jscm` *Generate Simulated Data for the Joint Scale-Change Model (JSCM)*

Description

Generates recurrent-event and terminal-event data under an AFT-type joint scale-change model using `simGSC`. Ported from `Data_gen_reReg()`.

Usage

```
generate_data_jscm(n, p, scenario = 1, b = 4)
```

Arguments

`n`	Integer. Number of subjects.
`p`	Integer. Number of covariates.
`scenario`	Integer. Scenario (1, 2, 3, or other).
`b`	Numeric. Upper bound of the censoring uniform distribution (default 4).

Details

In the JSCM convention:

- `alpha` governs the recurrence process.
- `beta` governs the terminal (death) process.

Covariates are drawn from `Uniform(-1, 1)`. A gamma frailty with shape = 4, scale = 1/4 is used. Censoring times are `Uniform(0, b)`.

Value

A list with components:

data Object returned by `simGSC` (a data frame with recurrent-event structure).

alpha_true True alpha (recurrence) coefficients.

beta_true True beta (terminal) coefficients.

Examples

```
dat <- generate_data_jscm(n = 30, p = 10, scenario = 1)
head(dat$data)
dat$alpha_true
dat$beta_true
```

plot.swjm_cv *Plot Cross-Validation Results*

Description

Plots the cross-validated estimating-equation score norms versus $\log(\lambda)$, with separate lines for the readmission and death components. A vertical dashed line marks the lambda that minimizes the combined norm. The top axis shows the total number of active variables along the path.

Usage

```
## S3 method for class 'swjm_cv'
plot(x, log_lambda = TRUE, ...)
```

Arguments

x	An object of class "swjm_cv".
log_lambda	Logical. If TRUE (default) the x-axis is $\log(\lambda)$; otherwise raw λ.
...	Currently unused.

Value

Invisibly returns x.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
cv  <- cv_stagewise(dat$data, model = "jfm", penalty = "coop",
                    max_iter = 200)
plot(cv)
```

plot.swjm_path *Plot a Stagewise Coefficient Path*

Description

Produces a glmnet-style plot of coefficient trajectories versus $\log(\lambda)$ along the stagewise regularization path. Two panels are drawn by default: one for the readmission sub-model (alpha) and one for the death sub-model (beta). The top axis of each panel shows the number of active variables at that lambda.

Usage

```
## S3 method for class 'swjm_path'
plot(
  x,
  log_lambda = TRUE,
  which = c("both", "readmission", "death"),
  col = NULL,
  ...
)
```

Arguments

x	An object of class "swjm_path".
log_lambda	Logical. If TRUE (default), the x-axis is $\log(\lambda)$; otherwise raw λ.
which	Character. Which sub-model(s) to plot: "both" (default), "readmission", or "death".
col	Optional vector of colors, one per covariate. Recycled if shorter than p.
...	Currently unused.

Value

Invisibly returns x.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
fit <- stagewise_fit(dat$data, model = "jfm", penalty = "coop",
                     max_iter = 200)
plot(fit)
```

---

plot.swjm_pred *Plot Predicted Survival Curves and Predictor Contributions*

Description

Produces a figure for a "swjm_pred" object. The layout depends on the model:

Usage

```
## S3 method for class 'swjm_pred'
plot(
  x,
  which_subject = 1L,
  which_process = c("both", "readmission", "death"),
```

```
  threshold = 0,
  ...
)
```

Arguments

x	An object of class "swjm_pred".
which_subject	Integer. Index of the subject to highlight (default 1).
which_process	Character. Which sub-model(s) to plot: "both" (default), "readmission", or "death".
threshold	Non-negative numeric. Only predictors whose absolute subject-specific contribution exceeds this value are shown in the bar chart. The default 0 suppresses variables with exactly zero contribution (i.e., unselected variables whose coefficient is zero). Increase threshold to focus on the most impactful predictors.
...	Currently unused.

Details

JFM (four panels):

1. Readmission-free survival curves for all subjects, with the selected subject highlighted.
2. Bar chart of readmission predictor contributions $\hat{\alpha}_j z_{ij}$ (log-hazard scale).
3. Death-free survival curves with the selected subject highlighted.
4. Bar chart of death predictor contributions $\hat{\beta}_j z_{ij}$.

JSCM (four panels):

1. Recurrent-event survival curves (AFT model) with the selected subject highlighted. The panel title shows the total acceleration factor $e^{\hat{\alpha}^\top z_i}$.
2. Bar chart of recurrence log time-acceleration contributions $\hat{\alpha}_j z_{ij}$: positive = events sooner, negative = later.
3. Death-free survival curves with the selected subject highlighted, total acceleration factor $e^{\hat{\beta}^\top z_i}$ in the title.
4. Bar chart of terminal-event log time-acceleration contributions $\hat{\beta}_j z_{ij}$.

In all cases bars represent *subject-specific* contributions (coefficient $\times$ covariate value), not bare coefficients, so the display correctly reflects how much each predictor shifts the log-hazard (JFM) or log time-acceleration (JSCM) for this particular subject.

Value

Invisibly returns x.

Examples

```
dat  <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
cv   <- cv_stagewise(dat$data, model = "jfm", penalty = "coop",
                     max_iter = 100)
newz <- matrix(rnorm(30), nrow = 3, ncol = 10)
pred <- predict(cv, newdata = newz)
plot(pred, which_subject = 2)
plot(pred, which_subject = 2, threshold = 0.05)
```

`predict.swjm_cv` *Predict from a Cross-Validated Joint Model Fit*

Description

Computes subject-specific predictions from a cross-validated `swjm_cv` fit. The output differs by model:

Usage

```
## S3 method for class 'swjm_cv'
predict(object, newdata, times = NULL, ...)
```

Arguments

`object`	An object of class `"swjm_cv"`.
`newdata`	A numeric matrix or data frame of covariate values. Must have p columns named `x1`, …, `xp` (if a data frame) or exactly p columns (if a matrix).
`times`	Numeric vector of evaluation times (JFM only). If `NULL`, the observed event times from the training data are used.
`...`	Currently unused.

Details

JFM — returns survival curves for both processes using the Breslow cumulative baseline hazards. For subject i with covariate vector z_i:

$$S_{\text{re}}(t \mid z_i) = \exp\!\big(-\hat{\Lambda}_0^r(t)\, e^{\hat{\alpha}^\top z_i}\big), \quad S_{\text{de}}(t \mid z_i) = \exp\!\big(-\hat{\Lambda}_0^d(t)\, e^{\hat{\beta}^\top z_i}\big).$$

JSCM — returns survival curves for both processes using a Nelson-Aalen baseline on the accelerated time scale. For subject i:

$$S_{\text{re}}(t \mid z_i) = \exp\!\big(-\hat{\Lambda}_0^r(t\, e^{\hat{\alpha}^\top z_i})\big), \quad S_{\text{de}}(t \mid z_i) = \exp\!\big(-\hat{\Lambda}_0^d(t\, e^{\hat{\beta}^\top z_i})\big).$$

The linear predictor $\hat{\alpha}^\top z_i$ is a log time-acceleration factor: the recurrence process for subject i runs on a time axis scaled by $e^{\hat{\alpha}^\top z_i}$ relative to the baseline. Each predictor contributes $\hat{\alpha}_j z_{ij}$ to this log-scale factor, so $e^{\hat{\alpha}_j z_{ij}}$ is the multiplicative factor on the time scale attributable to predictor j alone. Values greater than 1 accelerate events (shorter times); values less than 1 decelerate them (longer times).

Value

An object of class `"swjm_pred"`, a list with:

S_re Matrix of readmission-free survival probabilities (rows = subjects, columns = `times`).

S_de Matrix of death-free survival probabilities.

times Numeric vector of evaluation times.

lp_re Linear predictors for readmission ($\hat{\alpha}^\top z_i$). For JSCM this is the log time-acceleration for the recurrence process.

lp_de Linear predictors for death ($\hat{\beta}^\top z_i$). For JSCM this is the log time-acceleration for the terminal process.

time_accel_re (JSCM only) $e^{\hat{\alpha}^\top z_i}$: the multiplicative factor by which the recurrence time axis is scaled relative to baseline. `NULL` for JFM.

time_accel_de (JSCM only) Analogous time-acceleration factor for the terminal process. `NULL` for JFM.

contrib_re Matrix of per-predictor contributions $\hat{\alpha}_j z_{ij}$ (rows = subjects, columns = covariates). For JFM these are log-hazard contributions; for JSCM they are log time-acceleration contributions.

contrib_de Analogous matrix for the terminal process.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
cv  <- cv_stagewise(dat$data, model = "jfm", penalty = "coop",
                    max_iter = 100)
newz <- matrix(rnorm(30), nrow = 3, ncol = 10)
pred <- predict(cv, newdata = newz)
plot(pred)

dat_jscm <- generate_data(n = 50, p = 10, scenario = 1, model = "jscm")
cv_jscm  <- cv_stagewise(dat_jscm$data, model = "jscm", penalty = "coop",
                         max_iter = 500)
pred_jscm <- predict(cv_jscm, newdata = newz)
plot(pred_jscm)
```

`stagewise_fit` *Fit a Stagewise Regularization Path*

Description

Unified interface for stagewise variable selection in joint models of recurrent events and terminal events. Dispatches to model-specific implementations for the Joint Frailty Model (JFM) or Joint Scale-Change Model (JSCM).

Usage

```
stagewise_fit(
  data,
  model = c("jfm", "jscm"),
  penalty = c("coop", "lasso", "group"),
  eps = NULL,
  max_iter = NULL,
  pp = NULL,
  estimate_frailty = FALSE,
  standardize = TRUE
)
```

Arguments

data	A data frame in recurrent-event format with columns id, t.start, t.stop, event, status, and covariate columns x1, ..., xp.
model	Character. Either "jfm" or "jscm".
penalty	Character. One of "coop" (cooperative lasso), "lasso", or "group" (group lasso).
eps	Numeric. Step size for the stagewise update. If NULL, uses adaptive step size.
max_iter	Integer. Maximum number of stagewise iterations.
pp	Integer. Early-stopping block size: algorithm checks every pp iterations if fewer than 3 unique coordinates were updated.
estimate_frailty	Logical. For JFM only: if TRUE, estimates the frailty variance and uses the Kalbfleisch et al. (2013) frailty weights $w_i(t)$ in the estimating equations. If FALSE (default), uses unit weights (simplified model without frailty).
standardize	Logical. If TRUE (default), covariates are divided by their standard deviations before fitting and coefficients are rescaled back to the original scale. For the JFM with time-varying covariates, the SD is computed across all rows (all time points and subjects). For the JSCM with time-invariant covariates, the SD is computed across subjects only.

Value

An object of class "swjm_path", a list with components:

k Number of iterations performed.

theta Matrix of coefficient paths (2p rows by k+1 columns).

lambda Numeric vector of penalty parameter approximations at each iteration.

norm Numeric vector of penalty norm values along the path.

model Character, the model used.

penalty Character, the penalty used.

Examples

```
dat <- generate_data(n = 50, p = 10, scenario = 1, model = "jfm")
fit <- stagewise_fit(dat$data, model = "jfm", penalty = "coop",
                     max_iter = 100)
fit
```

Index