Package: palsr (via r-universe)

July 2, 2026

Type Package

Title Projected Actor Locations for Spatial Interaction Modeling

Version 0.1.0

Description Implements the Projected Actor Locations (PALS) method for spatial modeling of dyadic interactions between geographically mobile actors, as described in Kim, Liu and Desmarais (2023) <doi:10.1017/psrm.2022.6>. PALS applies exponential-smoothing weights to the spatiotemporal histories of a focal actor and its interaction partners (``alters'') to project the location of future interactions. The package provides projection, maximum-similarity parameter estimation by minimizing great-circle (Haversine) prediction error, nonparametric bootstrap with multiple-imputation (Rubin's Rules) pooling, dyadic distance covariate construction, visualization, and a simulated example dataset of subnational conflict.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports Rcpp, stats

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), knitr, rmarkdown, mice, ggplot2

URL https://github.com/bdesmarais/palsr

BugReports https://github.com/bdesmarais/palsr/issues

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Bruce A. Desmarais [aut, cre], Sangyeon Kim [aut], Howard Liu [aut]

Maintainer Bruce A. Desmarais <bruce.desmarais@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-01 10:00:13 UTC

RemoteUrl https://github.com/cran/palsr

RemoteRef HEAD

RemoteSha 5b39da2aa690779cc914f1e6aff5ce704d6d6fbd

Contents

`bootstrap_pals` *Nonparametric bootstrap for PALS estimates and projections*

Description

Quantifies uncertainty in PALS parameter estimates and in projected actor locations by resampling events with replacement and re-estimating the model on each bootstrap replicate, following Kim, Liu and Desmarais (2023). Each replicate yields a parameter vector and (optionally) a set of Projected Actor Locations; the collection of replicate PAL sets can be treated as multiple imputations and pooled with Rubin's Rules (see `pool_rubin()`).

Usage

```
bootstrap_pals(
  events,
  R = 50,
  model = c("four", "one"),
  predict_time = NULL,
  actors = NULL,
  seed = NULL,
```

```
  ...
)
```

Arguments

events	A pal_events object.
R	Number of bootstrap replicates (default 50; the paper uses 10).
model	"four" or "one" (passed to estimate_pals).
predict_time	Optional Date (or vector of Dates). When supplied, every replicate also projects all actors at these times, so the spread of projected coordinates across replicates is available for confidence regions and pooling.
actors	For projection, which actors to project (default: all in events).
seed	Optional integer seed; replicate r uses seed + r so the whole run is reproducible.
...	Further arguments passed to estimate_pals (e.g. aggregate, alter_weight, method, control).

Details

Resampling is over rows of events (the nonparametric event bootstrap). Duplicated events are kept as ordinary repeated events. Replicates whose optimizer fails to converge are retained but flagged via the convergence column of estimates.

Value

An object of class pals_boot with components:

estimates An R-row data.frame of replicate parameter estimates.

estimate The point estimate on the full sample (an estimate_pals fit).

projections If predict_time was given, a data.frame of projected lon/lat for every actor-time-replicate combination; otherwise NULL.

R, model, call Bookkeeping.

Methods: print(), summary() (bootstrap SEs / percentile intervals), and coef() (the full-sample point estimate).

See Also

estimate_pals(), pool_rubin().

Examples

```
ev <- simulate_conflict_events(n_actors = 8, n_events = 200, seed = 1)
bt <- bootstrap_pals(ev, R = 10, model = "one", seed = 1)
summary(bt)
```

`estimate_pals` *Estimate PALS parameters*

Description

Estimates the PALS smoothing parameters by minimizing the mean great-circle (Haversine) distance between observed event locations and the locations predicted from each event's preceding history ("marching forward": every prediction uses only events strictly earlier than the event being predicted).

Usage

```
estimate_pals(
  events,
  fit_events = NULL,
  model = c("four", "one"),
  start = NULL,
  method = NULL,
  aggregate = c("mean", "sum"),
  alter_weight = c("normalized", "legacy"),
  eps = 0.01,
  radius = 6371.0088,
  cutoff = c("day", "month", "year"),
  control = list()
)
```

Arguments

`events`	A pal_events object providing the actor histories.
`fit_events`	Optional `data.frame` of target events to fit against (needs `actor1`, `actor2`, `time`, `lon`, `lat`). Defaults to all of `events`; events with no usable history contribute nothing and are ignored.
`model`	`"four"` (full: focal + alter histories, estimates `alpha`, `beta`, `gamma`, `eta`) or `"one"` (focal-only; estimates `alpha`, with `pi = 0`).
`start`	Optional numeric starting vector on the optimizer's scale (`c(gamma, eta, log_alpha, log_beta)` for the four-parameter model, `log_alpha` for the one-parameter model). Sensible defaults are used if `NULL`.
`method`	Optimizer method passed to stats::optim (`"Nelder-Mead"` for the four-parameter model, `"Brent"` for the one-parameter model by default).
`aggregate`	`"mean"` (default) or `"sum"` of per-event distances.
`alter_weight, eps, cutoff`	See project_pal.
`radius`	Sphere radius for the Haversine objective (km).
`control`	A list of control parameters for stats::optim.

Value

An object of class `pals_fit` with components `params` (estimated pals_params), `model`, `objective` (minimized mean/sum distance), `n_used` (events contributing), `convergence`, `optim` (raw optimizer output), `events`, `settings`, and `call`. Methods: `print()`, `summary()`, `coef()`, `predict()`, `plot()`.

See Also

`project_pals()`, `predict_event_locations()`, `bootstrap_pals()`.

Examples

```
ev  <- simulate_conflict_events(n_actors = 10, n_events = 300, seed = 1)
fit <- estimate_pals(ev, model = "one")
fit
coef(fit)
```

`haversine` *Great-circle (Haversine) distance*

Description

Vectorized great-circle distance between longitude/latitude points. Arguments are recycled to a common length, so any may be length 1.

Usage

```
haversine(lon1, lat1, lon2, lat2, radius = 6371.0088)
```

Arguments

`lon1`, `lat1`, `lon2`, `lat2`	Numeric vectors of coordinates in decimal degrees.
`radius`	Sphere radius in the desired output units. Defaults to the mean Earth radius, `6371.0088` km, so distances are returned in kilometres.

Value

A numeric vector of distances. `NA` in any coordinate gives `NA`.

Examples

```
haversine(0, 0, 0, 1)          # ~111 km per degree of latitude
haversine(7.4, 9.1, 8.5, 12.0) # Abuja-ish to Kano-ish
```

`nigeria_acled` *Subnational conflict events in Nigeria*

Description

Real dyadic conflict events in Nigeria, bundled from the replication archive for the authors' study, Kim, Liu and Desmarais (2023). Each row is a recorded interaction between two actors at a known date and location; this is the data on which the PALS method was developed and validated.

Usage

```
nigeria_acled
```

Format

A pal_events object (a `data.frame` subclass) with 1,549 rows and 5 columns:

actor1 Character name of the first actor in the dyad.

actor2 Character name of the second actor in the dyad.

time `Date` of the event.

lon Event longitude (decimal degrees).

lat Event latitude (decimal degrees).

Details

These data are part of the publicly available replication materials for that study and can be downloaded directly from the Harvard Dataverse at doi:10.7910/DVN/NLWWPE.

Source

Public replication archive for Kim, Liu and Desmarais (2023), Harvard Dataverse, doi:10.7910/DVN/NLWWPE.

References

Kim, S., Liu, H., and Desmarais, B. A. (2023). Spatial modeling of dyadic geopolitical interactions between moving actors. *Political Science Research and Methods*, 11(3), 633-644. doi:10.1017/psrm.2022.6

Examples

```
data(nigeria_acled)
nigeria_acled

fit <- estimate_pals(nigeria_acled, model = "one")
coef(fit)
```

`nigeria_sim` *Simulated subnational conflict events (Nigeria-like)*

Description

A deterministic, seeded simulation of dyadic interaction events among 25 mobile actors over 2000-2016, with the qualitative spatiotemporal structure that the PALS method targets: actors drift slowly through space and interact preferentially with nearby actors. It is produced by `simulate_conflict_events()` and is used in the package examples, tests, and vignette so that they run without any external dependency. It complements nigeria_acled, the bundled real-data example.

Usage

```
nigeria_sim
```

Format

A pal_events object (a `data.frame` subclass) with 1500 rows and 5 columns:

actor1 Character id of the first actor in the dyad.

actor2 Character id of the second actor in the dyad.

time `Date` of the event.

lon Event longitude (decimal degrees).

lat Event latitude (decimal degrees).

Source

Generated by `data-raw/nigeria_sim.R` via `simulate_conflict_events(n_actors = 25, n_events = 1500, years = 2000:2016, seed = 20230101)`.

Examples

```
data(nigeria_sim)
nigeria_sim

fit <- estimate_pals(nigeria_sim, model = "one")
coef(fit)
```

pal_distance *Dyadic distance between Projected Actor Locations*

Description

Builds the dyadic distance covariate used to model interaction likelihood: the Haversine distance between the two actors' Projected Actor Locations.

Usage

```
pal_distance(
  events,
  dyads,
  params,
  transform = c("none", "log"),
  offset = 0.01,
  alter_weight = c("normalized", "legacy"),
  eps = 0.01,
  cutoff = c("day", "month", "year")
)
```

Arguments

events	A pal_events object.
dyads	A data.frame with columns actor1, actor2, time.
params	A pals_params or fitted estimate_pals object.
transform	"none" (default) returns distance in km; "log" returns log(distance + offset), as used for interstate-conflict-style specifications.
offset	Offset added before logging (default 0.01).
alter_weight, eps, cutoff	See project_pal.

Value

dyads augmented with pal_distance (and, for transform = "log", pal_log_distance).

Examples

```
ev  <- simulate_conflict_events(n_actors = 8, n_events = 200, seed = 1)
fit <- estimate_pals(ev, model = "one")
dy  <- data.frame(actor1 = "G01", actor2 = "G02",
                  time = as.Date("2012-12-01"))
pal_distance(ev, dy, fit)
```

pal_events *Construct a validated dyadic-event table*

Description

pal_events() builds the core data object used throughout **palsr**: a table of dyadic interaction events, each involving two actors at a known time and location. Projected Actor Locations are computed from these histories.

Usage

```
pal_events(
  data,
  actor1 = "actor1",
  actor2 = "actor2",
  time = "time",
  lon = "lon",
  lat = "lat",
  drop_self = TRUE
)
```

Arguments

data	A data.frame with one row per dyadic event.
actor1, actor2	Column names (length-1 character) identifying the two actors involved in each event. The pair is treated as unordered.
time	Column name of the event time. Must be a Date, or coercible to one via as.Date().
lon, lat	Column names of the event longitude and latitude, in decimal degrees.
drop_self	Logical; drop events whose two actors are identical (default TRUE).

Details

Longitudes must lie in [-180, 180] and latitudes in [-90, 90]. Rows with missing actor, time, or coordinate values are dropped with a message.

Value

An object of class pal_events (a data.frame subclass) with canonical columns actor1, actor2, time, lon, lat, sorted by time.

Examples

```
df <- data.frame(
  from = c("A", "A", "B"),
  to   = c("B", "C", "C"),
  when = as.Date(c("2001-01-01", "2001-06-01", "2002-01-01")),
```

```
  x    = c(7.1, 8.0, 7.5),
  y    = c(9.0, 9.4, 10.1)
)
ev <- pal_events(df, actor1 = "from", actor2 = "to",
                 time = "when", lon = "x", lat = "y")
ev
```

`pals_params` *Create a PALS parameter set*

Description

A lightweight container for the four PALS smoothing parameters. Use it to project actor locations with known parameters (e.g. values reported in a paper), without estimating them from data.

Usage

```
pals_params(alpha, beta = 0, gamma = 0, eta = 0, model = c("four", "one"))
```

Arguments

`alpha`	Time-decay of the focal actor's own event history (`>= 0`). Larger values down-weight older events more steeply.
`beta`	Time-decay of the alters' event histories (`>= 0`). Ignored in the one-parameter model.
`gamma`	Intercept of the logistic mixing weight π. Higher values place more weight on the alters' average location. Ignored in the one-parameter model.
`eta`	Slope of the logistic mixing weight on the event-count ratio. Ignored in the one-parameter model.
`model`	Either `"four"` (full model: focal + alter histories) or `"one"` (focal-only; $\pi = 0$, so only `alpha` is used).

Details

The mixing weight is $\pi = \text{plogis}(\gamma + \eta v)$, where $v = (n_i/n_k)^{1/\sqrt{n_k}}$ compares the number of focal events (n_i) with the number of alter events (n_k). The projected location is $(1-\pi)$ times the recency-weighted mean of the focal actor's own past event locations plus π times the recency-weighted mean of its alters' locations. See `project_pals()` and the package vignette.

Value

An object of class `pals_params`.

Examples

```
p <- pals_params(alpha = 0.9, beta = 0.2, gamma = -10, eta = -10)
p
pals_params(alpha = 0.9, model = "one")
```

`pool_rubin` *Pool estimates across imputations with Rubin's Rules*

Description

Combines per-imputation point estimates and variances of a scalar quantity into a single pooled estimate with a variance that accounts for both within- and between-imputation uncertainty (Rubin, 1987). Use it to pool estimands computed on each bootstrap/imputation replicate of `bootstrap_pals()` — for example a regression coefficient from a dyadic model fit on each replicate's PAL distances.

Usage

```
pool_rubin(estimates, variances, df = FALSE, dfcom = Inf)
```

Arguments

`estimates`	Numeric vector of per-imputation point estimates Q_j.
`variances`	Numeric vector of per-imputation variances U_j (the squared standard errors), the same length as `estimates`.
`df`	Logical; if `TRUE`, also return the Barnard-Rubin adjusted degrees of freedom and a corresponding two-sided p-value for `H0: Q = 0`. Default `FALSE` reproduces the source code's normal-based pooling.
`dfcom`	Complete-data degrees of freedom, used only when `df = TRUE` (default `Inf`, the large-sample limit).

Details

With m imputations,

$$\bar{Q} = \frac{1}{m}\sum_j Q_j, \quad \bar{U} = \frac{1}{m}\sum_j U_j, \quad B = \frac{1}{m-1}\sum_j (Q_j - \bar{Q})^2,$$

and total variance $T = \bar{U} + (1 + 1/m)B$. The fraction of missing information is $(1 + 1/m)B/T$. When `df = TRUE`, the Barnard-Rubin (1999) small-sample degrees of freedom are used.

Value

A one-row `data.frame` with the pooled estimate `qbar`, within-imputation variance `ubar`, between-imputation variance `b`, total variance `t`, standard error `se`, fraction of missing information `fmi`, and (if `df = TRUE`) `df` and `p.value`.

References

Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.

Barnard, J. and Rubin, D. B. (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948-955.

See Also

`bootstrap_pals()`.

Examples

```
# Five imputations of a coefficient and its variance.
q <- c(1.10, 0.95, 1.20, 1.05, 0.98)
u <- c(0.04, 0.05, 0.045, 0.038, 0.052)
pool_rubin(q, u)
pool_rubin(q, u, df = TRUE, dfcom = 100)
```

`predict.pals_fit` *Project locations from a fitted PALS model*

Description

Project locations from a fitted PALS model

Usage

```
## S3 method for class 'pals_fit'
predict(
  object,
  newdata = NULL,
  predict_time = NULL,
  type = c("pal", "event"),
  actors = NULL,
  ...
)
```

Arguments

`object`	A `pals_fit` from estimate_pals.
`newdata`	Optional pal_events (for `type = "pal"`) or target `data.frame` with `actor1`, `actor2`, `time` (for `type = "event"`). Defaults to the fitted events.
`predict_time`	For `type = "pal"`, a `Date` (or vector) at which to project.
`type`	`"pal"` projects per-actor locations; `"event"` predicts dyadic event locations (mean of the two actors' PALs).
`actors`	For `type = "pal"`, which actors to project (default: all).
`...`	Unused.

Value

A `data.frame` of projections (see project_pals / predict_event_locations).

Examples

```
ev  <- simulate_conflict_events(n_actors = 8, n_events = 200, seed = 1)
fit <- estimate_pals(ev, model = "one")
predict(fit, predict_time = as.Date("2013-12-01"), type = "pal")[1:5, ]
```

`predict_event_locations` *Predict dyadic event locations*

Description

For each dyadic target (a pair of actors at a time), predicts the interaction location as the mean of the two actors' Projected Actor Locations. Optionally scores the prediction against an observed location.

Usage

```
predict_event_locations(
  events,
  targets,
  params,
  alter_weight = c("normalized", "legacy"),
  eps = 0.01,
  cutoff = c("day", "month", "year")
)
```

Arguments

`events`	A pal_events object supplying the histories.
`targets`	A `data.frame` with columns `actor1`, `actor2`, `time`, and optionally `lon`/`lat` giving the observed event location (for error scoring).
`params`	A pals_params or fitted estimate_pals object.
`alter_weight, eps, cutoff`	See project_pal.

Value

`targets` augmented with `pred_lon`, `pred_lat`, and (if observed `lon`/`lat` were supplied) `error_km`, the Haversine distance between predicted and observed locations. Predictions are `NA` when both actors lack usable history.

Examples

```
ev  <- simulate_conflict_events(n_actors = 10, n_events = 300, seed = 1)
fit <- estimate_pals(ev, model = "one")
tg  <- ev[ev$time > as.Date("2012-01-01"), ]
head(predict_event_locations(ev, tg, fit))
```

`project_pal` *Project the location of a single actor*

Description

Computes the Projected Actor Location (PAL) for one actor at one or more prediction times, given a parameter set.

Usage

```
project_pal(
  events,
  actor,
  predict_time,
  params,
  alter_weight = c("normalized", "legacy"),
  eps = 0.01,
  cutoff = c("day", "month", "year")
)
```

Arguments

`events`	A pal_events object.
`actor`	The focal actor id (length-1 character).
`predict_time`	A `Date` (or vector of `Dates`) at which to project. Only events strictly earlier than each prediction time are used.
`params`	A pals_params object or a fitted estimate_pals object.
`alter_weight`	`"normalized"` (default) uses the paper's normalized alter weights; `"legacy"` reproduces the original replication code (an un-normalized sum of alter coordinates). See `DECISIONS.md`.
`eps`	Numerical offset inside each age weight (default `0.01`).
`cutoff`	History granularity: `"day"` (default) uses events strictly before `predict_time`; `"month"` uses events before the 1st of that month; `"year"` uses events before Jan 1 of that year (the source's Dec-1 yearly convention). Ages are always measured from `predict_time`.

Value

A `data.frame` with one row per prediction time and columns `actor`, `time`, `lon`, `lat`, `n_focal`, `n_alter`, `has_history`. `lon`/`lat` are `NA` when the actor has no prior events.

Examples

```
ev <- simulate_conflict_events(n_actors = 8, n_events = 200, seed = 1)
p  <- pals_params(alpha = 0.9, model = "one")
project_pal(ev, actor = "G01", predict_time = as.Date("2010-12-01"), params = p)
```

`project_pals` *Project locations for multiple actors*

Description

Computes Projected Actor Locations for a set of actors at one or more prediction times (an actor-by-time grid).

Usage

```
project_pals(
  events,
  actors = NULL,
  predict_time,
  params,
  alter_weight = c("normalized", "legacy"),
  eps = 0.01,
  cutoff = c("day", "month", "year")
)
```

Arguments

`events`	A pal_events object.
`actors`	Character vector of actor ids. Defaults to all actors in `events`.
`predict_time`	A `Date` or vector of `Dates`.
`params`	A pals_params object or a fitted estimate_pals object.
`alter_weight`	`"normalized"` (default) uses the paper's normalized alter weights; `"legacy"` reproduces the original replication code (an un-normalized sum of alter coordinates). See `DECISIONS.md`.
`eps`	Numerical offset inside each age weight (default `0.01`).
`cutoff`	History granularity: `"day"` (default) uses events strictly before `predict_time`; `"month"` uses events before the 1st of that month; `"year"` uses events before Jan 1 of that year (the source's Dec-1 yearly convention). Ages are always measured from `predict_time`.

Value

A `data.frame` with columns `actor`, `time`, `lon`, `lat`, `n_focal`, `n_alter`, `has_history` (one row per actor-time combination).

Examples

```
ev <- simulate_conflict_events(n_actors = 10, n_events = 300, seed = 1)
p  <- pals_params(alpha = 0.9, beta = 0.2, gamma = -10, eta = -10)
pal <- project_pals(ev, predict_time = as.Date("2010-12-01"), params = p)
head(pal)
```

`simulate_conflict_events` *Simulate dyadic conflict events between moving actors*

Description

Generates a synthetic dataset of dyadic interaction events with the qualitative structure PALS is designed for: a set of armed-group-like actors, each following a slowly drifting spatial trajectory, that interact preferentially with nearby actors. Useful for examples, tests, and the package vignette. The geographic frame approximates Nigeria, echoing the application in Kim, Liu and Desmarais (2023).

Usage

```
simulate_conflict_events(
  n_actors = 30,
  n_events = 2000,
  years = 2000:2016,
  drift = 0.18,
  jitter = 0.25,
  decay = 2,
  bbox = c(2.7, 14.7, 4, 13.9),
  seed = NULL
)
```

Arguments

`n_actors`	Number of actors (default 30).
`n_events`	Number of dyadic events (default 2000).
`years`	Integer vector of years to span (default `2000:2016`).
`drift`	Standard deviation (decimal degrees per year) of each actor's directional drift; larger values make actors more mobile (default `0.18`).
`jitter`	Standard deviation (decimal degrees) of event-location noise around the midpoint of the two actors' current locations (default `0.25`).
`decay`	Spatial interaction scale (degrees): partners are chosen with probability proportional to `exp(-distance / decay)`, so nearby actors interact more (default `2`).
`bbox`	Bounding box `c(lon_min, lon_max, lat_min, lat_max)` for actor home locations (default approximately Nigeria).
`seed`	Optional integer seed for reproducibility.

Value

A pal_events object with columns `actor1`, `actor2`, `time`, `lon`, `lat`.

Examples

```
ev <- simulate_conflict_events(n_actors = 12, n_events = 400, seed = 42)
ev
summary(ev)
```

Index