# Package: simITS (via r-universe)

August 27, 2024

**Type** Package

**Title** Analysis via Simulation of Interrupted Time Series (ITS) Data

**Version** 0.1.1

**Description** Uses simulation to create prediction intervals for post-policy outcomes in interrupted time series (ITS) designs, following Miratrix (2020) <arXiv:2002.05746>. This package provides methods for fitting ITS models with lagged outcomes and variables to account for temporal dependencies. It then conducts inference via simulation, simulating a set of plausible counterfactual post-policy series to compare to the observed post-policy series. This package also provides methods to visualize such data, and also to incorporate seasonality models and smoothing and aggregation/summarization. This work partially funded by Arnold Ventures in collaboration with MDRC.

**License** GPL-3

**Depends** dplyr, R (>= 2.10), rlang

**Suggests** arm, ggplot2, knitr, plyr, purrr, rmarkdown, stats, testthat (>= 2.1.0), tidyr

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**Repository** https://lmiratrix.r-universe.dev

**RemoteUrl** https://github.com/lmiratrix/simits

**RemoteRef** HEAD

**RemoteSha** 6a4c3cb91f0357ca6d1ba21c5dc063566088d4b3

## Contents

---

## `add_lagged_covariates` *Augment dataframe with lagged covariates*

---

### Description

Take outcome and a list of covariates and add new columns with lagged versions. Assumes rows of dataframe are in time ascending order. Lagged outcome canonically called 'lag.outcome'. Covariates 'lag.XXX'.

### Usage

```
add_lagged_covariates(dat, outcomename, covariates = NULL)
```

### Arguments

| | |
|---|---|
| `dat` | The dataframe |
| `outcomename` | The outcome of interest (string) |
| `covariates` | The covariates to lag along with the outcome. This can be either of two things. First, it can be a list of string names. Covariates can also be a function with a "lags" attribute with the listed covariates (as returned by, e.g., make_fit_season_model) (which is a list of string names). NULL if no covariates other than outcome should be lagged. |

### Value

Augmented dataframe with lagged covariates as new columns. Will clobber old columns if the names (of form "lag.XXXX") conflict.

### Examples

```
data( "newjersey" )
newjersey = add_lagged_covariates(newjersey, "n.warrant", c("sin.m","cos.m" ) )
head( newjersey[ c( "n.warrant", "sin.m", "lag.outcome", "lag.sin.m" ) ] )
```

---

## adjust_data *Adjust an outcome time series based on the group weights.*

---

### Description

Reweight the components of a series to match target weights for several categories. This is a good preprocessing step to adjust for time-varying covariates such as changing mix of case types.

### Usage

```
adjust_data(
  dat,
  outcomename = "Y",
  timename = "time",
  groupname = "G",
  Nname = "N",
  pi_star,
  is_count = FALSE,
  include_aggregate = FALSE,
  covariates = NULL
)
```

### Arguments

| | |
|---|---|
| `dat` | Dataframe of data. Requires an N column of total cases represented in each row. |
| `outcomename` | Name of column that has the outcome to calculated adjusted values for. |
| `groupname` | Name of categorical covariate that determines the groups. |
| `Nname` | Name of column in dat that contains total cases (this is the name of the variable used to generate the weights in pi_star). |
| `pi_star` | The target weights. Each time point will have its groups re-weighted to match these target weights. |
| `is_count` | Indicator of whether outcome is count data or a continuous measure (this impacts how aggregation is done). |
| `include_aggregate` | Include aggregated (unadjusted) totals in the output as well. |
| `covariates` | Covariates to be passed to aggregation (list of string variable names). |

## Value

Dataframe of adjusted data.

## Examples

```
data( "meck_subgroup" )
head( meck_subgroup )
pis = calculate_group_weights( "category", Nname="n.cases",
                               meck_subgroup, t_min=0, t_max= max( meck_subgroup$month ) )
pis

agg = aggregate_data( meck_subgroup,
                      outcomename="pbail", groupname="category", Nname="n.cases",
                      is_count=FALSE,
                      rich = TRUE, covariates = NULL )
head( agg )


adjdat = adjust_data( meck_subgroup, "pbail", "category", "n.cases", pis, include_aggregate=TRUE )
head( adjdat )
```

---

`aggregate_data` *Aggregate grouped data*

---

## Description

This will take a dataframe with each row being the outcomes, etc., for a given group for a given time point and aggregate those groups for each time point.

## Usage

```
aggregate_data(
  dat,
  outcomename = "Y",
  timename = "time",
  groupname = "G",
  Nname = "N",
  is_count = FALSE,
  rich = TRUE,
  covariates = NULL
)
```

## Arguments

| | |
|---|---|
| `dat` | Dataframe with one row for each time point and group that we are going to post stratify on. This dataframe should also have an column with passed name "Nname" indicating the number of cases that make up each given row. It should have a column "timename" for the time. |

| | |
|---|---|
| `outcomename` | String name of the outcome variable in dat. |
| `groupname` | Name of the column that has the grouping categorical variable |
| `Nname` | Name of variable holding the counts (weight) in each group. |
| `is_count` | If TRUE the data are counts, and should be aggregated by sum rather than by mean. |
| `rich` | If TRUE, add a bunch of extra columns with proportions of the time point that are each group and so forth. |
| `covariates` | group-invariant covariates to preserve in the augmented rich dataframe. These are not used in this method for any calculations. Pass as list of column names of dat |

## Value

Dataframe of aggregated data, one row per time point If rich=TRUE many extra columns with further information.

## Examples

```
data( "meck_subgroup" )
head( meck_subgroup )
pis = calculate_group_weights( "category", Nname="n.cases",
                             meck_subgroup, t_min=0, t_max= max( meck_subgroup$month ) )
pis

agg = aggregate_data( meck_subgroup,
                      outcomename="pbail", groupname="category", Nname="n.cases",
                      is_count=FALSE,
                      rich = TRUE, covariates = NULL )
head( agg )


adjdat = adjust_data( meck_subgroup, "pbail", "category", "n.cases", pis, include_aggregate=TRUE )
head( adjdat )
```

---

`aggregate_simulation_results`

*Test a passed test statistic on the simulated data*

---

## Description

This method is used to look at summary statistics such as average impact post-policy, and see how the predictive distribution compares to the observed.

### Usage

```
aggregate_simulation_results(
  orig.data,
  predictions,
  outcomename = "Y",
  timename = "time",
  summarizer = calculate_average_outcome,
  ...
)
```

### Arguments

| | |
|---|---|
| `orig.data` | The raw data (dataframe) |
| `predictions` | The results from process_outcome_model. |
| `outcomename` | Outcome to use. |
| `summarizer` | A function to calculate some summary quantity, Default: calculate_average_outcome |
| `...` | Extra arguments passed to the summarizer function. |

### Value

List of length two, with first item being the observed value of the test statistic and the second being a numeric vector representing the emperical reference distribution.

### Examples

```
predictions = process_outcome_model( "pbail", mecklenberg,
                                    t0=0, R = 5,
                                    summarize = FALSE, smooth=FALSE )
sstat = aggregate_simulation_results( orig.data = mecklenberg, outcomename = "pbail",
                                     predictions = predictions, time_points = 1:18 )
sstat$t
sstat$t.obs
```

## calculate_average_outcome

*Summary function for summarize.simulation.results*

### Description

Given a set of simulation runs, estimate average impact over range of time points.

### Usage

```
calculate_average_outcome(res, outcomename, timename, time_points = 1:54, ...)
```

### Arguments

| | |
|---|---|
| `res` | Dataframe of a single series (simulated or otherwise) |
| `outcomename` | Name of outcome in res |
| `time_points` | Which time points to average over, Default: 1:18 |
| `...` | Other parameters (ignored) |

### Value

Single number (in this case mean of given time points)

### See Also

See `aggregate_simulation_results` for how this function would be used.

### Examples

```
data( mecklenberg )
calculate_average_outcome( mecklenberg, "pbail", time_points=1:24 )
calculate_average_outcome( mecklenberg, "pbail", time_points = 1:18 )
```

## calculate_group_weights

*Calculate proportion of subgroups across time*

### Description

Calculate overall proportion of cases in each group that lie within a given interval of time defined by t_min and t_max.

### Usage

```
calculate_group_weights(groupname, dat, t_min, t_max = Inf, Nname = "N")
```

### Arguments

| | |
|---|---|
| `groupname` | Name of the column that has the grouping categorical variable |
| `dat` | Dataframe with one row for each time point and group that we are going to post stratify on. This dataframe should also have an column with passed name "Nname" indicating the number of cases that make up each given row. It should have a column "timename" for the time. |
| `t_min` | The start time point to aggregate cases over. |
| `t_max` | The final time point (default is last time point). |
| `Nname` | Name of variable holding the counts (weight) in each group. |

## Value

Dataframe of each group along with overall average group weight in the specified timespan.

## Examples

```
data( "meck_subgroup" )
head( meck_subgroup )
pis = calculate_group_weights( "category", Nname="n.cases",
                               meck_subgroup, t_min=0, t_max= max( meck_subgroup$month ) )
pis

agg = aggregate_data( meck_subgroup,
                      outcomename="pbail", groupname="category", Nname="n.cases",
                      is_count=FALSE,
                      rich = TRUE, covariates = NULL )
head( agg )


adjdat = adjust_data( meck_subgroup, "pbail", "category", "n.cases", pis, include_aggregate=TRUE )
head( adjdat )
```

---

`extrapolate_model` *Extrapolate pre-policy data to post-policy era*

---

## Description

This function takes a fitted model and uses it to make post-policy predictions by simulating data.

## Usage

```
extrapolate_model(
  M0,
  dat,
  outcomename = "Y",
  timename = "time",
  t0 = 0,
  R = 400,
  summarize = FALSE,
  smooth = FALSE,
  smoother = smooth_series,
  full_output = FALSE,
  fix_parameters = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `M0` | The fit model |
| `dat` | Dataframe with data being analyzed. |
| `outcomename` | Outcome of interest (name of column). |
| `timename` | Name of the time variable (name of column). |
| `t0` | Last pre-policy timepoint |
| `R` | Number of replications |
| `summarize` | Boolean, TRUE means collapse all simulated trajectories into single aggregate. FALSE means return all paths. |
| `smooth` | Boolean. TRUE means fit a smoother to the trajectories and look at distribution of smoothed trajectories. FALSE means look at raw data treajectories. |
| `smoother` | Function to do smoothing, if smoothing set to TRUE. Default is smooth_series() |
| `full_output` | TRUE means smoother returns residuals as well as smoothed series. |
| `fix_parameters` | Keep the parameters in the model M0 as fixed; do not add parameter uncertainty. |
| `...` | Extra arguments to be passed to smoother (e.g, bandwidth). |

## Value

Dataframe with columns corresponding to the simulations. If summarize=TRUE, one row per time point in original data. If FALSE, all the details of all the runs are returned.

## See Also

`process_outcome_model` for wrapper function for this method that is easier to use.

## Examples

```
data("mecklenberg" )
mecklenberg = add_lagged_covariates( mecklenberg, "pbail"  )
mecklenberg.pre = dplyr::filter( mecklenberg, month <= 0 )
M0 = fit_model_default( mecklenberg.pre, "pbail" )
res = extrapolate_model( M0, "pbail", mecklenberg, 0, 1,
                         smooth=TRUE)
tail( res )
```

---

`fit_model_default` *Default ITS model*

---

## Description

This fits the model ‘outcomename ~ lag.outcome + time‘, with no covariates.

## Usage

```
fit_model_default(dat, outcomename, timename = "time", lagless = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `dat` | Dataframe of pre-policy data to fit model to. Needs a "time" column |
| `outcomename` | Outcome of interest |
| `lagless` | Boolean, include the lagged outcome, or not? |
| `...` | Extra arguments passed to the lm() call. |

### Value

A fit model (a ‘lm‘ object from a ‘lm()‘ call) from fitting a simple regression of outcome onto time and lagged time.

### Examples

```
mecklenberg = add_lagged_covariates(mecklenberg, "pbail")
meck.pre = filter( mecklenberg, month <= 0 )
mod = fit_model_default( meck.pre, "pbail", "month", lagless = TRUE )
summary( mod )
mod = fit_model_default( meck.pre, "pbail", "month", lagless = FALSE )
summary( mod )
```

## `generate_fake_data` *Make fake data for testing purposes.*

### Description

Defaults have heavy seasonality, and an extra bump in impact kicks in at 12 months post-policy.

### Usage

```
generate_fake_data(
  t_min = -40,
  t_max = 9,
  t0 = 0,
  rho = 0.5,
  sd.omega = 1,
  coef_line = c(20, 0.05),
  coef_q = c(1, 0, -1, 0),
  coef_temp = 0.1,
  coef_sin = c(0, 0),
  coef_tx = c(0, 0.25, 5)
)
```

### Arguments

| | |
|---|---|
| `t_min` | Index of first month |
| `t_max` | Index of last month |
| `t0` | Last pre-policy time point |
| `rho` | Autocorrelation |
| `sd.omega` | Standard deviation of the true residual |
| `coef_line` | Intercept and slope of the main trendline (list of 2). |
| `coef_q` | Coefficients for the four quarters (list of 4). |
| `coef_temp` | Coefficient for temperature. |
| `coef_sin` | Coefficents for sin and cos features (list of 2) |
| `coef_tx` | Coefficient for treatment post-policy (list of 3, initial offset, initial slope, additional slope past 12 months). Treatment is a piecewise linear function. |

### Value

A `tibble` having `month` , `temperature` , `sin.m` , `cos.m` , `Q1`, `Q2` , `Q3`, `Q4`, `post` , `Ystr0` , `Ystr` , `Y`

### Examples

```
fdat = generate_fake_data(-100,100, rho = 0.95, coef_q=c(0,0,0,0), coef_temp = 0)
plot( fdat$month, fdat$Y, type="l" )
fdat2 = generate_fake_data(-100, 100, rho = 0.0, coef_q=c(0,0,0,0), coef_temp = 0)
plot( fdat$month, fdat2$Y, type="l" )
```

---

## `generate_fake_grouped_data`

*A fake DGP with time varying categorical covariate for illustrating the code.*

---

### Description

This code makes synthetic grouped data that can be used to illustrate benefits of post stratification.

### Usage

```
generate_fake_grouped_data(
  t_min,
  t0,
  t_max,
  method = c("complex", "linear", "jersey")
)
```

### Arguments

| | |
|---|---|
| `t_min` | Index of first time point |
| `t0` | last pre-policy timepoint |
| `t_max` | Index of last time point |
| `method` | Type of post-stratification structure to generate (three designs of 'complex', 'linear' and 'jersey' were originally concieved of when designing simulation studies with different types of structure). |

### Value

Dataframe of fake data, with one row per group per time period.

### Examples

```
fdat = generate_fake_grouped_data(t_min=-5,t_max=10, t0 = 0)
table( fdat$time )
table( fdat$type )
```

---

## `make_envelope_graph` *Make envelope style graph with associated smoothed trendlines*

---

### Description

This method builds a ggplot object with the trendline and prediction envelope. It can be customized after the fact by adding more ggplot layers via normal ggplot "+" syntax.

### Usage

```
make_envelope_graph(envelope, t0, ylab = "Y", xlab = "month")
```

### Arguments

| | |
|---|---|
| `envelope` | The result of a 'process_outcome_model()' call, i.e. dataframe with columns of original data, imputed data and, potentially, smoothed data. |
| `t0` | Last pre-policy timepoint. Will draw vertical line here. |
| `ylab` | Y label of plot |
| `xlab` | X label of plot |

### Value

Returns (does not yet display) a ggplot plot object containing the time series along with extrapolation and prediction envelope. This plot can be augmented and changed via standard ggplot commands.

### See Also

The `ggplot2` package.

## Examples

```
data( "mecklenberg" )
t0 = 0
envelope = process_outcome_model( "pbail", mecklenberg,
                                  t0=t0, R = 10,
                                  summarize = TRUE, smooth=FALSE )
make_envelope_graph(envelope, t0=t0, ylab = "Proportion given bail") +
  ggplot2::labs( title="Sample ITS plot")
data( "mecklenberg" )
t0 = 0
envelope = process_outcome_model( "pbail", mecklenberg,
                                  t0=t0, R = 10,
                                  summarize = TRUE, smooth=FALSE )
make_envelope_graph(envelope, t0=t0, ylab = "Proportion given bail") +
  ggplot2::labs( title="Sample ITS plot")
```

---

## `make_fit_season_model` *Make a fit_model function that takes a seasonality component*

---

### Description

This method returns a function that will fit a model both with and without lagged outcomes.

### Usage

```
make_fit_season_model(formula, no_lag = NULL)
```

### Arguments

| | |
|---|---|
| `formula` | Formula specifying seasonality. No outcome or time needed. |
| `no_lag` | Formula specifying additional covariates that should be included, but without lag (usually used due to colinearity of lagged outcomes, such as with a sin and cos component). |

### Details

You hand it a formula object specifying the seasonality, e.g., " ~ Q2 + Q3 + Q4", if you have quarterly season effects. This method assumes you want models with a linear time component as well, and will add that and an intercept in automatically.

It gives you a function back, that you can use to analyze data.

### Value

Callable function. See make_fit_model.

### See Also

make_fit_model for the type of function this method will generate.

## Examples

```
data( "newjersey")
modF = make_fit_season_model( ~ temperature, timename = "month" )
newjersey = add_lagged_covariates( newjersey, "n.warrant", covariates = c("temperature") )
modF( newjersey, "n.warrant" )
```

---

## `make_many_predictions` *Generate a collection of raw counterfactual trajectories*

---

### Description

Given a fit linear model 'fit0', generate R prediction series starting at t0. This takes model uncertainty into account by pulling from the pseudo-posterior of the model parameters (from Gelman and Hill arm package).

### Usage

```
make_many_predictions(fit0, dat, R, outcomename, timename, t0)

make_many_predictions_plug(fit0, dat, R, outcomename, timename, t0)
```

### Arguments

| | |
|---|---|
| `fit0` | The fit linear model to simulate from. |
| `dat` | A dataframe with the covariates needed by the model fit0 for both pre and post-policy time points |
| `R` | Number of series to generate. |
| `outcomename` | The name of the column in dat which is our outcome. |
| `t0` | Last time point of pre-policy. Will start predicting at t0+1. |

### Value

A data.frame with the collection of predicted series, one row per time point per replicate (so will have R*nrow(dat) rows).

### Functions

- `make_many_predictions_plug()`: This version makes multiple predictions using estimated parameters without additional uncertainty. This takes point estimates from the fit model as fixed parameters. WARNING: This method will not capture true uncertainty as it is not taking parameter uncertainty into account.

### References

The 'arm' package, see https://cran.r-project.org/package=arm

Also see Gelman, A., & Hill, J. (2007). Data analysis using regression and multilevelhierarchical models (Vol. 1). New York, NY, USA: Cambridge University Press.

### Examples

```
data("mecklenberg" )
mecklenberg = add_lagged_covariates( mecklenberg, "pbail"  )
mecklenberg.pre = dplyr::filter( mecklenberg, month <= 0 )
M0 = fit_model_default( mecklenberg.pre, "pbail" )
res = make_many_predictions( M0, dat=mecklenberg, outcome="pbail", t0=0, R=2 )
tail( res )
```

---

## `make_model_smoother` *Make a smoother that fits a model and then smooths residuals*

---

### Description

This helper function gives back a function that takes the resulting simulation data from a single iteration of the simulation, and fits 'fit_model' to it, smoothes the residuals, and puts the predictions from 'fit_model' back.

### Usage

```
make_model_smoother(fit_model, covariates)
```

### Arguments

| | |
|---|---|
| `fit_model` | A function that takes data, fits a linear model, and returns the fit model. This function needs an option to include (or not) lagged covariates. |
| `covariates` | A dataframe with all covariates needed in the model fitting defined by fit_model. |

### Details

This can be used to build smoothers that smooth using models other than the model being used for extrapolation (e.g., a model without temperature).

Resulting functions have the following parameters: 'res' (the data), 't0' (start time), 'outcomename', 'post.only' flag (for smoothing only post data or not), and 'smooth_k', a tuning parameter for degree of smoothing.

### Value

A smoother function that can be passed to the smoothing routines. This function is of the form listed above.

### Examples

```
data( "newjersey")
modA = make_fit_season_model( ~ temperature )
modB = make_fit_season_model( ~ sin.m + cos.m )
newjersey = add_lagged_covariates( newjersey, "n.warrant",
                                   covariates = c("sin.m", "cos.m", "temperature") )
smoother = make_model_smoother( fit_model = modA, covariates = newjersey )
```

```
class(smoother)

# Pass made function to process_outcome_model()
envelope = process_outcome_model( "n.warrant", newjersey, t0=-8, R = 1,
                                  summarize = TRUE, smooth=TRUE,
                                  smoother = smoother, smooth_k = 11,
                                  fit.model = modB )
```

---

mecklenberg *Mecklenberg PSA Reform Data*

---

## Description

Monthly aggregate outcomes of various measures of interest from Mecklenberg. See MDRC Report.

## Usage

```
mecklenberg
```

## Format

A data frame with 54 rows and 10 variables:

`month` integer Month, with 0 being month of policy implementation.

`karr` integer Total count of arrests.

`pbail` double Proportion of cases in a given month assigned bail (or outright detention).

`pptrel` double Proportion of cases assigned to pretrial supervised release.

`pror` double Proportion of cases released on own recognizance.

`pb4c` double Proportion of cases assigned to money bail (alternate coding from pbail, above).

`avg_days_initial` double Average number of days spent detained before release due to bail, case resolution, etc.

`avg_t2d` double Average time to case resolution (in days).

`pstint7` double Proportion detained longer than 7 days.

`pstint30` double Proportion detained longer than 30 days.

## `meck_subgroup` *Mecklenberg data by subgroup of charge type*

### Description

Mecklenberg data that gives proportion of different charge categories of cases given bail (by month).

### Usage

```
meck_subgroup
```

### Format

A data frame with 144 rows and 5 variables:

`month` integer Month, with 0 being month of policy implementation.

`n.cases` integer Number of cases of that subgroup for that month

`n.bail` interger Total number of cases given bail for that subgroup for that month

`pbail` double Proportion of new cases in given subgroup in that month assigned bail

`category` character Category of group (charge type).

## `newjersey` *New Jersey PSA Reform aggregate data*

### Description

Montly aggregate counts of arrests of different types in New Jersey.

### Usage

```
newjersey
```

### Format

A data frame with 106 rows and 11 variables:

`month` integer Index of month.

`sin.m` double cos of month number

`cos.m` double sin of month number

`M12` integer Month number

`Q1` integer Indicator of 1st quarter.

`Q2` integer Indicator of 2nd quarter.

`Q3` integer Indicator of 3rd quarter.

`Q4` integer Indicator of 4th quarter.

`n.warrant` double Number of warrant arrests

`n.summons` double Number of summons arrests

`n` double Total number of arrests

`temperature` double Average temperature in New Jersey that month.

---

## `process_outcome_model` *Generate an ITS extrapolation simulation.*

---

### Description

This is the primary function to use to use this approach on a given dataset.

### Usage

```
process_outcome_model(
  dat,
  outcomename = "Y",
  timename = "time",
  t0 = 0,
  R = 400,
  summarize = FALSE,
  smooth = FALSE,
  smoother = NULL,
  fit_model = fit_model_default,
  covariates = NULL,
  plug_in = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `dat` | Dataframe with a column for time. The time column is assumed to be a sequence of integer values. |
| `outcomename` | Name of column in dat containing the observed outcomes. |
| `timename` | Name of column in dat containing the time points. |
| `t0` | Last pre-policy timepoint |
| `R` | Number of simulated pre-policy extrapolations to generate. |
| `summarize` | Summarise the series? (TRUE/FALSE) |
| `smooth` | Smooth the series? (TRUE/FALSE) |
| `smoother` | Function to smooth residuals, if smoothing set to TRUE. If NULL, will dynamically make a model smoother based on the fit_model method if covariates are passed. Otherwise it will use the simple smoother on the outcomes. |

| | |
|---|---|
| `fit_model` | The function used to fit the model to simulate from. (This model could be a seasonality model. Default is simple linear model with no covariates.) |
| `covariates` | Vector of covariate names of all covariates used in the passed model function fit_model. If null, will attempt to get list of covariates form the "lags" attribute of the passed 'fit_model'. |
| `plug_in` | Use the estimated parameters as fixed and do not include extra uncertainty of parameter estimation in the simulation. (Not recommended as it destroys inference.) |
| `...` | Extra arguments to be passed to ‘extrapolate_model()‘ |

## Details

Take a given outcome variable, fit an ITS model, use it to extrapolate R plusible trajectories, and then using these trajectories, generate final impact results by averaging (if summarize is set to TRUE).

This function is basically a wrapper for ‘extrapolate_model()‘ with some extra calls to ‘make_model_smoother()‘ to prepare, in the case of smoothing, and adding on a summary trend via ‘generate_Ybars()‘ in the case of summarizing.

## Value

If summarize=TRUE, A dataframe with several columns of interest and one row per time point of data. The columns are Ymin and Ymax, the limits of the envelope, 'range', the range of the envelope, 'SE', the standard deviation of the trajectories at that time point, ‘Ysmooth‘ the median smoothed value at that time point (if smoothing), ‘Ystar‘ the median unsmoothed value at that time point (regardless of smooth flag), ‘Y‘, the observed outcome, ‘Ysmooth1‘, the smoothed observed outcomes, and ‘Ybar‘ the predicted outcome given the model with no autoregressive aspect.

If summarize=FALSE, a dataframe of all the raw series generated.

## See Also

The core internal function that this method is a wrapper for is `extrapolate_model`.

## Examples

```
data( "mecklenberg" )
t0 = 0
envelope = process_outcome_model( "pbail", mecklenberg,
                                  t0=t0, R = 10,
                                  summarize = TRUE, smooth=FALSE )
make_envelope_graph(envelope, t0=t0, ylab = "Proportion given bail") +
  ggplot2::labs( title="Sample ITS plot")
```

## simITS — simITS *package overview*

### Description

Analysis via Simulation of Interrupted Time Series

### Details

This package is based on the backbone analytic code for the analyses in, e.g., Redcross et al. (2019) or Golub et al. (2019). See companion paper Miratrix (2020) for technical discussion of the overall approach.

Broadly, this package provides methods for fitting Interrupted Time Series models with lagged outcomes and variables to account for temporal dependencies. It then conducts inference via simulation, simulating a set of plausible counterfactual post-policy series to compare to the observed post-policy series. This package provides methods to visualize such data, and also to incorporate seasonality models and smoothing and aggregation/summarization. See the vignette for a guide of how to conduct such analyses.

### References

Redcross, C., Henderson, B., Valentine, E. & Miratrix, L. (2019). Evaluation of pretrial justice system reforms that use the public safety assessment: Effects in Mecklenburg County, North Carolina. Technical report, MDRC (link)

Golub, C. A., Redcross, C., Valentine, E., & Miratrix, L. (2019). Evaluation of pretrial justice system reforms that use the public safety assessment: Effects of New Jersey's criminal justice reform. Technical report, MDRC. (link)

Miratrix, L. (2020). Using Simulation to Analyze Interrupted Time Series Designs (link)

## smooth_residuals — *Smooth residuals after model fit*

### Description

Smooth a series by fitting the model to the data, smoothing the residuals, and then putting the model predictions back.

### Usage

```
smooth_residuals(
  res,
  t0 = 0,
  outcomename = "Y",
  timename = "time",
  post.only = TRUE,
  smooth_k = SMOOTH_K,
  fit_model = fit_model_default,
  covariates = res,
  full_output = FALSE
)
```

## Arguments

| | |
|---|---|
| `res` | A dataframe with a 'timename' column and an 'outcomename' column (which is the column that will be smoothed). |
| `t0` | last pre-policy timepoint |
| `outcomename` | String name of the outcome variable in dat. Defaut is "Y". |
| `timename` | Name of the time column. Default is "time". |
| `post.only` | If TRUE fit model and smooth post-policy only. WHY fit model on post-policy data only? Because this will make sure the fixed pre-policy does not dominate too much? We are focusing on post-policy so we want a good fitting model for that so we can get our residuals as "white noise" as possible before smoothing. |
| `smooth_k` | A rough proxy for the number of observations to primarily consider to kernal weight in the neighborhood of each timepoint (this is a bandwidth, and the loess smoother gets smooth_k / n as a span value). We want to smooth with an absolute bandwidth, not one as function of how long the time series is. |
| `fit_model` | A function that takes data, fits a linear model, and returns the fit model. This function needs an option to include (or not) lagged covariates. |
| `covariates` | A dataframe with all covariates needed in the model fitting defined by fit_model. |
| `full_output` | If TRUE give back pieces for diagnostics of smoothing process. |

## Details

Use loess smoother on complete series of residuals including original data pre-policy and synthetic data post policy (i.e., smooth the entire plausible series).

## Value

A numeric vector of the smoothed residuals. If full_output=TRUE return a dataframe with several other columns: ‘resid‘, the residuals based on Ystar and the model, ‘residStar‘ the smoothed residuals, 'Ybar.sm' the structural predictions of the model used for smoothing. Here the smoothed values will be 'Ysmooth'.

## See Also

See `smooth_series` for a more vanilla version that smooths without the model fitting step.

## Examples

```
data( "newjersey" )
smooth = smooth_series( newjersey, outcomename = "n.warrant", t0= -8,
                        smooth_k = 30,
                        post.only = FALSE)
plot( newjersey$month, newjersey$n.warrant )
lines( newjersey$month, smooth, col="red" )

mod =  make_fit_season_model( ~ temperature )
newjersey = add_lagged_covariates( newjersey, outcomename = "n.warrant",
                                   covariates = c("temperature") )

smooth = smooth_residuals( newjersey, outcomename = "n.warrant", t0=-8,
                           smooth_k = 30,
                           post.only = FALSE,
                           fit_model = mod )
plot( newjersey$month, newjersey$n.warrant )
lines( newjersey$month, smooth, col="red" )
```

---

`smooth_series` *Smooth a series using a static loess smoother*

---

## Description

Use loess smoother on complete series of residuals including original data pre-policy and synthetic data post policy (i.e., smooth the entire plausible series).

## Usage

```
smooth_series(
  res,
  outcomename = "Y",
  timename = "time",
  t0 = 0,
  smooth_k = SMOOTH_K,
  post.only = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `res` | A dataframe with a ’timename’ column and an ’outcomename’ column (which is the column that will be smoothed). |
| `outcomename` | String name of the outcome variable in dat. Defaut is "Y". |
| `timename` | Name of the time column. Default is "time". |
| `t0` | last pre-policy timepoint |

| | |
|---|---|
| `smooth_k` | A rough proxy for the number of observations to primarily consider to kernal weight in the neighborhood of each timepoint (this is a bandwidth, and the loess smoother gets smooth_k / n as a span value). We want to smooth with an absolute bandwidth, not one as function of how long the time series is. |
| `post.only` | If TRUE fit model and smooth post-policy only. WHY fit model on post-policy data only? Because this will make sure the fixed pre-policy does not dominate too much? We are focusing on post-policy so we want a good fitting model for that so we can get our residuals as "white noise" as possible before smoothing. |
| `...` | Extra arguments (not used in this function). |

## Details

This method takes several parameters it does not use, to maintain compatability with smooth_residuals.

## Value

An updated version of the 'res' dataframe with ‘Ysmooth‘, the smoothed predictions of the original Ystar outcome. Also includes 'Ystar' the original sequence to be smoothed.

## Examples

```
data( "newjersey" )
smooth = smooth_series( newjersey, outcomename = "n.warrant", t0= -8,
                        smooth_k = 30,
                        post.only = FALSE)
plot( newjersey$month, newjersey$n.warrant )
lines( newjersey$month, smooth, col="red" )

mod =  make_fit_season_model( ~ temperature )
newjersey = add_lagged_covariates( newjersey, outcomename = "n.warrant",
                                   covariates = c("temperature") )

smooth = smooth_residuals( newjersey, outcomename = "n.warrant", t0=-8,
                           smooth_k = 30,
                           post.only = FALSE,
                           fit_model = mod )
plot( newjersey$month, newjersey$n.warrant )
lines( newjersey$month, smooth, col="red" )
```

# Index