Package 'unmconf'

September 10, 2024

Type Package

Title Modeling with Unmeasured Confounding

Version 1.0.0

Maintainer David Kahle <david@kahle.io>

Description Tools for fitting and assessing Bayesian multilevel regression
 models that account for unmeasured confounders.

SystemRequirements JAGS >= 4.3.0 (http://mcmc-jags.sourceforge.net)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 2.10), rjags

Imports stats, glue, janitor, coda

Suggests bayesplot, posterior, ggplot2, dplyr, tidyr, tibble, knitr,
 rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Ryan Hebdon [aut],
 James Stamey [aut] (<https://orcid.org/0000-0002-3787-6490>),
 David Kahle [aut, cre] (<https://orcid.org/0000-0002-9999-1558>),
 Xiang Zhang [aut]

Repository CRAN

Date/Publication 2024-09-09 22:00:02 UTC

Contents

helpers 2
runm 3
unmconf 5
unm_glm 5
unm_summary 14

Index **18**

`helpers` *Convert to Greek expressions*

Description

Convert to Greek expressions for plotting

Usage

```
expand_labels(labs)

greek_expander(s)

make_greek_coefs(mod)

drop_nulls(x)
```

Arguments

`labs`	A character vector of greek symbols of the form `ga_x` and `be_1`.
`s`	A character vector of Greek short hand codes, e.g. `"si"`.
`mod`	Output from `unm_glm()`.
`x`	Character vector.

Value

A character vector.

Examples

```
labs <- c("ga_1", "ga_treatment", "ga_x", "be_1",
  "be_treatment", "be_x", "la_u", "al_y", "si")
expand_labels(labs)
```

runm *Generate synthetic data*

Description

runm() generates synthetic data for use of modeling with unmeasured confounders. Defaults to the case of one unmeasured confounder present and fixed parameter values. Can be customized. Currently set up to have at most two unmeasured confounders to pair with unm_glm().

Usage

```
runm(
  n,
  type = "int",
  missing_prop = 0.8,
  response = "bin",
  response_param,
 response_model_coefs = c(int = -1, z1 = 0.5, z2 = 0.5, z3 = 0.5, u1 = 0.5, x = 0.5),
  treatment_model_coefs = c(int = -1, z1 = 0.5, z2 = 0.5, z3 = 0.5, u1 = 0.5),
  covariate_fam_list = list("norm", "bin", "norm"),
  covariate_param_list = list(c(mean = 0, sd = 1), prob = 0.3, c(0, 2)),
  unmeasured_fam_list = list("norm"),
  unmeasured_param_list = list(c(mean = 0, sd = 1))
)
```

Arguments

`n`	Number of observations. When `type = "int"`, `n` is a vector of length 1. When `type = "ext"`, `n` can either be a vector of length 1 or 2. For the case when `n` is of length 2, `n = (n_main, n_external)`, where `n_main` corresponds to the main study sample size and `n_external` corresponds to the external validation sample size. For the case when `n` is of length 1, `n` will be split evenly between main study and external validation observations, with the main study getting the additional observation when `n` is odd.
`type`	Type of validation source. Can be `"int"` for internal validation or `"ext"` for external validation. Defaults to `"int"`.
`missing_prop`	Proportion of missing values for internal validation scenario (i.e., when `type = "int"`).
`response`	`"norm"`, `"bin"`, `"pois"`, or `"gam"`. Defaults to `"bin"`.
`response_param`	Nuisance parameters for response type. For `"norm"`, the default standard deviation is 1. For `"gam"`, the default shape parameter is 2. For `"pois"`, an offset variable is added to the dataset that is uniformly distributed from 1 to 10.
`response_model_coefs`	A named vector of coefficients to generate data from the response model. This must include an intercept (`"int" = `), a coefficient for each covariate specified, a coefficient for each unmeasured confounder, and a treatment coefficient

("x" =). The coefficients for the covariates and treatment will be denoted with "beta[.]" and the unmeasured confounders with "lambda[.]".

`treatment_model_coefs`
: A named vector of coefficients to generate data from the treatment model. This must include an intercept ("int" =), a coefficient for each covariate specified, and a coefficient for each unmeasured confounder. The coefficients for the covariates and unmeasured confounders will be denoted with "eta[.]".

`covariate_fam_list`
: A list of either "norm" or "bin", where the length of the list matches the number of covariates in the model.

`covariate_param_list`
: A list of parameters for the respective distributions in `covariate_fam_list`, where the length of the list matches the length of `covariate_fam_list`.

`unmeasured_fam_list`
: A list of either "norm" or "bin", where the length of the list matches the number of unmeasured confounders in the model. This can be at most a length of 2 to pair with `unm_glm()`.

`unmeasured_param_list`
: A list of parameters for the respective distributions in `unmeasured_fam_list`, where the length of the list matches the length of `unmeasured_fam_list`.

Value

A `tibble`

Examples

```
runm(100)
runm(n = 100, type = "int", missing_prop = .75)
runm(n = 100, type = "int", missing_prop = .75) |> attr("params")
runm(100, type = "int", response = "norm")
runm(100, type = "int", response = "norm") |> attr("params")
runm(100, type = "int", response = "norm", response_param = 3) |> attr("params")
runm(100, type = "int", response = "gam")
runm(100, type = "int", response = "gam", response_param = 5) |> attr("params")
runm(100, type = "int", missing_prop = .5, response = "pois")

runm(n = 100, type = "ext")
runm(n = 100, type = "ext") |> attr("params")
runm(n = c(10, 10), type = "ext")
runm(100, type = "ext", response = "norm")
runm(100, type = "int", response = "norm", response_param = 3) |> attr("params")
runm(100, type = "ext", response = "gam")
runm(100, type = "ext", response = "pois")

runm(
  n = 100,
  type = "int",
  missing_prop = .80,
  response = "norm",
  response_param = c("si_y" = 2),
  response_model_coefs = c("int" = -1, "z" = .4,
                           "u1" = .75, "u2" = .75, "x" = .75),
  treatment_model_coefs = c("int" = -1, "z" = .4,
                            "u1" = .75, "u2" = .75),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm", "bin"),
  unmeasured_param_list = list(c(mean = 0, sd = 1), c(.3))
)

runm(
  n = c(20, 30),
  type = "ext",
  response = "norm",
  response_param = c("si_y" = 2),
  response_model_coefs = c("int" = -1, "z1" = .4, "z2" = .5, "z3" = .4,
                           "u1" = .75, "u2" = .75, "x" = .75),
  treatment_model_coefs = c("int" = -1, "z1" = .4, "z2" = .5, "z3" = .4,
                            "u1" = .75, "u2" = .75),
  covariate_fam_list = list("norm", "bin", "norm"),
  covariate_param_list = list(c(mean = 0, sd = 1), c(.3), c(0, 2)),
  unmeasured_fam_list = list("norm", "bin"),
  unmeasured_param_list = list(c(mean = 0, sd = 1), c(.3))
)
```

unmconf — *unmconf: Modeling with Unmeasured Confounding*

Description

Tools for fitting and assessing Bayesian multilevel regression models that account for unmeasured confounders.

unm_glm — *Fitting Multilevel Bayesian Regression Model with Unmeasured Confounders*

Description

`unm_glm()` fits a multilevel Bayesian regression model that accounts for unmeasured confounders. Users can input model information into `unm_glm()` in a similar manner as they would for the standard `stats::glm()` function, providing arguments like `formula`, `family`, and `data`. Results are stored as MCMC iterations.

Usage

```
unm_glm(
  form1,
  form2 = NULL,
  form3 = NULL,
  family1 = binomial(),
  family2 = NULL,
  family3 = NULL,
  data,
  n.iter = 2000,
  n.adapt = 1000,
  thin = 1,
  n.chains = 4,
  filename = tempfile(fileext = ".jags"),
  quiet = getOption("unm_quiet"),
  progress.bar = getOption("unm_progress.bar"),
  code_only = FALSE,
  priors,
  response_nuisance_priors,
  response_params_to_track,
  confounder1_nuisance_priors,
  confounder1_params_to_track,
  confounder2_nuisance_priors,
  confounder2_params_to_track,
  ...
)

jags_code(mod)

## S3 method for class 'unm_int'
print(x, digits = 3, ..., print_call = getOption("unm_print_call"))

## S3 method for class 'unm_int'
coef(object, ...)
```

Arguments

`form1`	The formula specification for the response model (Level I)
`form2`	The formula specification for the first unmeasured confounder model (Level II)
`form3`	The formula specification for the second unmeasured confounder model (Level III)
`family1`, `family2`, `family3`	The family object, communicating the types of models to be used for response (`form1`) and unmeasured confounder (`form2, form3`) models. See `stats::family()` for details
`data`	The dataset containing all variables (this function currently only supports a single dataset containing internally validated data)

n.iter	n.iter argument of rjags::coda.samples()
n.adapt	n.adapt argument of rjags::jags.model()
thin	thin argument of rjags::coda.samples()
n.chains	n.chains argument of rjags::jags.model()
filename	File name where to store jags code
quiet	The quiet parameter of rjags::jags.model(). Defaults to TRUE, but you can change it on a per-session basis with options(unm_quiet = FALSE).
progress.bar	The progress.bar parameter of rjags::update.jags(). Defaults to "none", but you can change it on a per-session basis with options(unm_progress.bar = "text").
code_only	Should only the code be created?
priors	Custom priors to use on regression coefficients, see examples.
response_nuisance_priors, confounder1_nuisance_priors, confounder2_nuisance_priors	JAGS code for the nuisance priors on parameters in a JAGS model (see examples)
response_params_to_track, confounder1_params_to_track, confounder2_params_to_track	Additional parameters to track when nuisance parameter priors are used (see examples)
...	Additional arguments to pass into rjags::jags.model(), such as inits
mod	The output of unm_glm()
x	Object to be printed
digits	Number of digits to round to; defaults to 3
print_call	Should the call be printed? Defaults to TRUE, but can be turned off with options("unm_print_call" = FALSE)
object	Model object for which the coefficients are desired

Value

(Invisibly) The output of rjags::coda.samples(), an object of class mcmc.list, along with attributes code containing the jags code used and file containing the filename of the jags code.

See Also

runm(), rjags::dic.samples()

Examples

```
# ~~ One Unmeasured Confounder Examples (II-Level Model) ~~


# normal response, normal confounder model with internally validated data
(df <- runm(20, response = "norm"))
```

```
(unm_mod <- unm_glm(
  form1 = y ~ x + z1 + z2 + z3 + u1,  family1 = gaussian(),
  form2 = u1 ~ x + z1 + z2 + z3,      family2 = gaussian(),
  data = df
))

## Not run:  # reduce cran check time

(unm_mod <- unm_glm(
  y ~ .,      family1 = gaussian(),
  u1 ~ . - y, family2 = gaussian(),
  data = df
))
glm(y ~ x + z1 + z2 + z3, data = df)
coef(unm_mod)

jags_code(unm_mod)
unm_glm(
  y ~ .,
  u1 ~ . - y,
  family1 = gaussian(),
  family2 = gaussian(),
  data = df, code_only = TRUE
)




# a normal-normal model - external validation
(df <- runm(c(10, 10), type = "ext", response = "norm"))

unm_glm(
  y ~ x + z1 + z2 + z3 + u1,  family1 = gaussian(),
  u1 ~ x + z1 + z2 + z3,      family2 = gaussian(),
  data = df
)



# setting custom priors
unm_glm(
  y ~ .,      family1 = gaussian(),
  u1 ~ . - y, family2 = gaussian(),
  data = df,
  code_only = TRUE
)

unm_glm(
  y ~ .,      family1 = gaussian(),
  u1 ~ . - y, family2 = gaussian(),
  data = df,
  code_only = FALSE,
  priors = c("lambda[u1]" = "dnorm(1, 10)"),
  response_nuisance_priors = "tau_{y} <- sigma_{y}^-2; sigma_{y} ~ dunif(0, 100)",
  response_params_to_track = "sigma_{y}",
  confounder1_nuisance_priors = "tau_{u1} <- sigma_{u1}^-2; sigma_{u1} ~ dunif(0, 100)",
  confounder1_params_to_track = "sigma_{u1}"
)


# turn progress tracking on
options("unm_progress.bar" = "text")



# more complex functional forms _for non-confounder predictors only_
# zero-intercept model
unm_glm(
  y ~ . - 1,
  u1 ~ . - y,
  family1 = gaussian(),
  family2 = gaussian(),
  data = df
)
glm(y ~ . - 1, data = df)

# polynomial model
unm_glm(
  y ~ x + poly(z1, 2) + u1,
  u1 ~ x + z1,
  family1 = gaussian(),
  family2 = gaussian(),
  data = df
)
glm(y ~ x + poly(z1, 2), data = df)

# interaction model
unm_glm(
  y ~ x*z1 + u1, family1 = gaussian(),
  u1 ~ x*z1,     family2 = gaussian(),
  data = df
)
glm(y ~ x*z1, data = df)



# a binomial-binomial model
(df <- runm(
  50,
  missing_prop = .75,
  response = "bin",
  unmeasured_fam_list = list("bin"),
  unmeasured_param_list = list(.5)
))
(unm_mod <- unm_glm(
  y ~ .,
  u1 ~ . - y,
  family1 = binomial(),
  family2 = binomial(),
  data = df
))
glm(y ~ . - u1, family = binomial(), data = df)



# a poisson-normal model
(df <- runm(
  25,
  response = "pois",
  response_model_coefs = c("int" = -1, "z" = .5, "u1" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5, "u1" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm"),
  unmeasured_param_list = list(c(0, 1))
))

(unm_mod <- unm_glm(
  y ~ x + z + u1 + offset(log(t)),
  u1 ~ x + z,
  family1 = poisson(),
  family2 = gaussian(),
  data = df
))
glm(y ~ x + z + offset(log(t)), family = poisson(), data = df)



# a poisson-binomial model
(df <- runm(
  25,
  response = "pois",
  response_model_coefs = c("int" = -1, "z" = .5, "u1" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5, "u1" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("bin"),
  unmeasured_param_list = list(.5)
))

(unm_mod <- unm_glm(
  y ~ x + z + u1 + offset(log(t)), family1 = poisson(),
  u1 ~ x + z,                      family2 = binomial(),
  data = df
))

glm(y ~ x + z + offset(log(t)), family = poisson(), data = df)
```

```
# a gamma-normal model
(df <- runm(
  25,
  response = "gam",
  response_model_coefs = c("int" = -1, "z" = .5, "u1" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5, "u1" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm"),
  unmeasured_param_list = list(c(0, 1))
))

(unm_mod <- unm_glm(
  y ~ x + z + u1, family1 = Gamma(),
  u1 ~ x + z,     family2 = gaussian(),
  data = df
))

glm(y ~ x + z, family = Gamma(link = "log"), data = df)



# a gamma-binomial model
(df <- runm(
  25,
  response = "gam",
  response_model_coefs = c("int" = -1, "z" = .5, "u1" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5, "u1" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("bin"),
  unmeasured_param_list = list(.5)
))

(unm_mod <- unm_glm(
  y ~ x + z + u1, family1 = Gamma(),
  u1 ~ x + z,     family2 = binomial(),
  data = df
))
print(df, n = 25)

glm(y ~ x + z, family = Gamma(link = "log"), data = df)



# the output of unm_glm() is classed jags output

(df <- runm(20, response = "norm"))
(unm_mod <- unm_glm(
  y ~ .,
  u1 ~ . - y,
  family1 = gaussian(),
  family2 = gaussian(),
  data = df)
)
class(unm_mod)
jags_code(unm_mod)
unm_glm(y ~ ., u1 ~ . - y, data = df, code_only = TRUE)






# visualizing output
library("ggplot2")
library("bayesplot"); bayesplot_theme_set(ggplot2::theme_minimal())
mcmc_hist(unm_mod, facet_args = list(labeller = label_parsed))
mcmc_hist(unm_mod)
mcmc_trace(unm_mod, facet_args = list(labeller = label_parsed))

# more extensive visualization with the tidyverse
mcmc_intervals(unm_mod, prob = .90) +
  geom_point(
    aes(value, name), data = tibble::enframe(attr(df, "params")),
    color = "red", fill = "pink", size = 4, shape = 21
  )


library("dplyr")
library("tidyr")
unm_mod %>%
  as.matrix() %>%
  as_tibble() %>%
  pivot_longer(everything(), names_to = "var", values_to = "val") %>%
  ggplot(aes("0", val)) +
  geom_jitter() +
  geom_point(
    aes("0", value), data = tibble::enframe(attr(df, "params"), name = "var"),
    color = "red", fill = "pink", size = 4, shape = 21
  ) +
  coord_flip() +
  facet_grid(var ~ ., scales = "free_y", labeller = label_parsed) +
  theme_bw() +
  theme(
    axis.title = element_blank(),
    axis.text.y = element_blank(), axis.ticks.y = element_blank(),
    strip.text.y = element_text(angle = 0)
  )


# getting draws out
(samps <- posterior::as_draws_df(unm_mod))
samps$`.chain`
samps$`.iteration`
samps$`.draw`
```

```
# implementation is variable-name independent
(df <- runm(100, response = "norm"))
df$ht <- df$y
df$age <- df$u1
df$biom <- df$x
(unm_mod <- unm_glm(
  ht ~ x + biom + age,
  age ~ x + biom,
  data = df,
  family1 = gaussian(),
  family2 = gaussian()
))
jags_code(unm_mod)

# ~~ Two Unmeasured Confounders Examples (III-Level Model) ~~
# a normal-normal-normal model - internal validation
(df <- runm(
  50,
  missing_prop = .75,
  response = "norm",
  response_model_coefs = c("int" = -1, "z" = .5,
                           "u1" = .5, "u2" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5,
                            "u1" = .5, "u2" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm", "norm"),
  unmeasured_param_list = list(c(0, 1), c(0, 1))
))
(unm_mod <- unm_glm(
  y ~ x + z + u1 + u2,
  u1 ~ x + z + u2,
  u2 ~ x + z,
  family1 = gaussian(),
  family2 = gaussian(),
  family3 = gaussian(),
  data = df
))

glm(y ~ x + z, data = df)
coef(unm_mod)

unm_glm(
  y ~ x + z + u1 + u2, family1 = gaussian(),
  u1 ~ x + z + u2,   family2 = gaussian(),
  u2 ~ x + z,        family3 = gaussian(),
  data = df,
  code_only = TRUE
)
```

```
# a normal-normal-normal model - external validation
(df <- runm(
  c(20, 20),
  type = "ext",
  response = "norm",
  response_model_coefs = c("int" = -1, "z" = .5,
                           "u1" = .5, "u2" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5,
                            "u1" = .5, "u2" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm", "norm"),
  unmeasured_param_list = list(c(0, 1), c(0, 1))
))
(unm_mod <- unm_glm(
  y ~ x + z + u1 + u2,  family1 = gaussian(),
  u1 ~ x + z + u2,      family2 = gaussian(),
  u2 ~ x + z,           family3 = gaussian(),
  data = df
))



# a binomial-binomial-binomial model - internal validation
(df <- runm(
  25,
  response = "bin",
  response_model_coefs = c("int" = -1, "z" = .5,
                           "u1" = .5, "u2" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5,
                            "u1" = .5, "u2" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("bin", "bin"),
  unmeasured_param_list = list(.5, .75)
))
unm_glm(y ~ x + z + u1 + u2, family1 = binomial(),
        u1 ~ x + z + u2,     family2 = binomial(),
        u2 ~ x + z,          family3 = binomial(),
        data = df,
        code_only = TRUE
)



## End(Not run)
```

`unm_summary` *Generate synthetic data*

Description

`unm_summary()` produces result summaries of the results from the model fitting function, `unm_glm()`. The table of results are summarized from the MCMC draws of the posterior distribution.

Usage

```
unm_summary(mod, data, quantiles = c(0.025, 0.5, 0.975))

unm_backfill(data, mod)

unm_dic(mod)
```

Arguments

`mod`	Output from `unm_glm()`.
`data`	The data `mod` was generated with.
`quantiles`	A numeric vector of quantiles.

Value

A `tibble`

Examples

```
# ~~ One Unmeasured Confounder Examples (II-Level Model) ~~

# normal response, normal confounder model with internally validated data
(df <- runm(20, response = "norm"))

(unm_mod <- unm_glm(
  y ~ x + z1 + z2 + z3 + u1, family1 = gaussian(),
  u1 ~ x + z1 + z2 + z3,     family2 = gaussian(),
  data = df
))

glm(y ~ x + z1 + z2 + z3, data = df)

coef(unm_mod)
jags_code(unm_mod)
unm_summary(unm_mod)
unm_summary(unm_mod, df) # true values known df

# impute missing values with model
unm_backfill(df, unm_mod)

## Not run:  # reduce cran check time

# a normal-normal model - external validation
(df <- runm(c(10, 10), type = "ext", response = "norm"))
```

```
(unm_mod <- unm_glm(
  y ~ x + z1 + z2 + z3 + u1,
  u1 ~ x + z1 + z2 + z3,
  family1 = gaussian(),
  family2 = gaussian(),
  data = df
))

unm_backfill(df, unm_mod)


# a binomial-binomial model
(df <- runm(
  50,
  missing_prop = .75,
  response = "bin",
  unmeasured_fam_list = list("bin"),
  unmeasured_param_list = list(.5)
))

(unm_mod <- unm_glm(
  y ~ .,
  u1 ~ . - y,
  family1 = binomial(),
  family2 = binomial(),
  data = df
))

glm(y ~ . - u1, family = binomial(), data = df)
unm_backfill(df, unm_mod)
unm_summary(unm_mod, df)

# computing the dic. penalty = effective number of parameters
unm_dic(unm_mod)
coef(unm_mod)
unm_backfill(df, unm_mod)


# ~~ Two Unmeasured Confounders Examples (III-Level Model) ~~
# a normal-normal-normal model - internal validation
(df <- runm(
  50,
  missing_prop = .75,
  response = "norm",
  response_model_coefs = c("int" = -1, "z" = .5,
                           "u1" = .5, "u2" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5,
                            "u1" = .5, "u2" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm", "norm"),
  unmeasured_param_list = list(c(0, 1), c(0, 1))
))
```

```
(unm_mod <- unm_glm(
  y ~ x + z + u1 + u2, family1 = gaussian(),
  u1 ~ x + z + u2, family2 = gaussian(),
  u2 ~ x + z, family3 = gaussian(),
  data = df
))
glm(y ~ x + z, data = df)
coef(unm_mod)
unm_summary(unm_mod)
unm_summary(unm_mod, df) # true values known df
unm_backfill(df, unm_mod)

# a normal-normal-normal model - external validation
(df <- runm(
  c(20, 20),
  type = "ext",
  response = "norm",
  response_model_coefs = c("int" = -1, "z" = .5,
                           "u1" = .5, "u2" = .5, "x" = .5),
  treatment_model_coefs = c("int" = -1, "z" = .5,
                            "u1" = .5, "u2" = .5),
  covariate_fam_list = list("norm"),
  covariate_param_list = list(c(mean = 0, sd = 1)),
  unmeasured_fam_list = list("norm", "norm"),
  unmeasured_param_list = list(c(0, 1), c(0, 1))
))

(unm_mod <- unm_glm(
  y ~ x + z + u1 + u2, family1 = gaussian(),
  u1 ~ x + z + u2, family2 = gaussian(),
  u2 ~ x + z, family3 = gaussian(),
  data = df
))
unm_backfill(df, unm_mod)

## End(Not run)
```

Index

coef.unm_int *(unm_glm)*, 5

drop_nulls *(helpers)*, 2

expand_labels *(helpers)*, 2

greek_expander *(helpers)*, 2

helpers, 2

jags_code *(unm_glm)*, 5

make_greek_coefs *(helpers)*, 2

print.unm_int *(unm_glm)*, 5

rjags::coda.samples(), *7*
rjags::dic.samples(), *7*
rjags::jags.model(), *7*
rjags::update.jags(), *7*
runm, 3
runm(), *3*, *7*

stats::family(), *6*
stats::glm(), *5*

unm_backfill *(unm_summary)*, 15
unm_dic *(unm_summary)*, 15
unm_glm, 5
unm_glm(), *2–5*, *7*, *15*
unm_summary, 14
unm_summary(), *15*
unmconf, 5
unmconf-package *(unmconf)*, 5