Package: pcvr (via r-universe)

November 7, 2024

Type Package

Title Plant Phenotyping and Bayesian Statistics

Version 1.1.1.0

Maintainer Josh Sumner <jsumner@danforthcenter.org>

Description Analyse common types of plant phenotyping data, provide a simplified interface to longitudinal growth modeling and select Bayesian statistics, and streamline use of 'PlantCV' output. Several Bayesian methods and reporting guidelines for Bayesian methods are described in Kruschke (2018) <doi:10.1177/2515245918771304>, Kruschke (2013) <doi:10.1037/a0029146>, and Kruschke (2021) <doi:10.1038/s41562-021-01177-7>.

Depends R (>= 3.5.0)

License GPL-2

Encoding UTF-8

Additional_repositories https://mc-stan.org/r-packages/

RoxygenNote 7.2.3

Imports FactoMineR, rlang, stats, utils, methods, data.table, ggplot2, ggridges, igraph, jsonlite, lme4, patchwork, extraDistr, parallel, bayestestR, viridis, mgcv, quantreg, nlme, splines, lmeSplines, scales, survival, car

Suggests knitr, rmarkdown, brms, flexsurv, curl, cmdstanr, rstan, caret, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/danforthcenter/pcvr, https://plantcv.org/, https://danforthcenter.github.io/pcvr/

BugReports https://github.com/danforthcenter/pcvr/issues

NeedsCompilation no

Author Josh Sumner [aut, cre]
(<https://orcid.org/0000-0002-3399-9063>), Jeffrey Berry [aut]
(<https://orcid.org/0000-0002-8064-9787>), Noah Fahlgren [rev]
(<https://orcid.org/0000-0002-5597-4537>), Donald Danforth
Plant Science Center [cph]

Repository CRAN

Date/Publication 2024-11-06 20:50:02 UTC

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libxml2-dev
zlib1g-dev

Contents

`awkHelper` *subset helper function for use reading in large data, called in pcv.sub.read*

Description

subset helper function for use reading in large data, called in pcv.sub.read

Usage

```
awkHelper(inputFile, filters, awk = NULL)
```

Arguments

`inputFile`	Path to csv file of plantCV output, should be provided internally in read.pcv
`filters`	filtering conditions, see read.pcv for details. Format as list("trait in area, perimeter", "other contains stringToMatch")
`awk`	Optional awk command to use instead.

Details

awkHelper attempts to make awk commands from human readable input. Currently when filters are supplied the input file has quotes removed by 'sed' then is piped into awk, so an equivalent command line statement may be: `sed 's/\"//g' pcvrTest2.csv | awk -F ',' '{ if (NR==1 || $18=="area") { print } }'`

Value

Returns a character string representing a unix style awk statement which is typically passed to `pipe` or used as a connection in `data.table::fread`.

Examples

```
tryCatch(
  { # in case offline
    link1 <- "https://gist.githubusercontent.com/seankross/"
   link2 <- "a412dfbd88b3db70b74b/raw/5f23f993cd87c283ce766e7ac6b329ee7cc2e1d1/mtcars.csv"
    file <- paste0(link1, link2)
    awkHelper(file, list("gear in 4, 3"), awk = NULL)
    awkHelper(file, "gear contains 3", awk = NULL)
   # note that to be filtered the file has to exist on your local system, this example only shows
    # the output of awkHelper, which would then be executed by read.pcv on a unix system
    awkHelper(file, list("gear in 4, 3"), awk = "existing_command")
  },
  error = function(e) {
    message(e)
  }
)
```

`barg` *Function to help fulfill elements of the Bayesian Analysis Reporting Guidelines.*

Description

The Bayesian Analysis Reporting Guidelines were put forward by Kruschke (https://www.nature.com/articles/s41562-021-01177-7) to aide in reproducibility and documentation of Bayesian statistical analyses that are sometimes unfamiliar to reviewers or scientists. The purpose of this function is to summarize goodness of fit metrics from one or more Bayesian models made by growthSS and fitGrowth. See details for explanations of those metrics and the output.

Usage

```
barg(fit, ss = NULL)
```

Arguments

`fit`	The brmsfit object or a list of brmsfit objects in the case that you split models to run on subsets of the data for computational simplicity.
`ss`	The growthSS output used to specify the model. If fit is a list then this can either be one growthSS list in which case the priors are assumed to be the same for each model or it can be a list of the same length as fit. Note that the only parts of this which are used are the `call$start` which is expected to be a call, `pcvrForm`, and `df` list elements, so if you have a list of brmsfit objects and no ss object you can specify a stand-in list. This can also be left NULL (the default) and posterior predictive plots and prior predictive plots will not be made.

Details

- **General**: This includes chain number, length, and total divergent transitions per model. Divergent transitions are a marker that the MCMC had something go wrong. Conceptually it may be helpful to think about rolling a marble over a 3D curve then having the marble suddenly jolt in an unexpected direction, something happened that suggests a problem/misunderstood surface. In practice you want extremely few (ideally no) divergences. If you do have divergences then consider specifying more control parameters (see brms::brm or examples for fitGrowth). If the problem persists then the model may need to be simplified. For more information on MCMC and divergence see the stan manual (https://mc-stan.org/docs/2_19/reference-manual/divergent-transitions).
- **ESS**: ESS stands for Effective Sample Size and is a goodness of fit metric that approximates the number of independent replicates that would equate to the same amount of information as the (autocorrelated) MCMC iterations. ESS of 1000+ is often considered as a pretty stable value, but more is better. Still, 100 per chain may be plenty depending on your applications and the inference you wish to do. One of the benefits to using lots of chains and/or longer chains is that you will get more complete information and that benefit will be shown by a larger ESS. This is separated into "bulk" and "tail" to represent the middle and tails of the posterior distribution, since those can sometimes have very different sampling behavior. A summary and the total values are returned, with the summary being useful if several models are included in a list for fit argument
- **Rhat**: Rhat is a measure of "chain mixture". It compares the between vs within chain values to assess how well the chains mixed. If chains did not mix well then Rhat will be greater than 1, with 1.05 being a broadly agreed upon cutoff to signify a problem. Running longer chains should result in lower Rhat values. The default in brms is to run 4 chains, partially to ensure that there is a good chance to check that the chains mixed well via Rhat. A summary and the total values are returned, with the summary being useful if several models are included in a list for fit argument
- **NEFF**: NEFF is the NEFF ratio (Effective Sample Size over Total MCMC Sample Size). Values greater than 0.5 are generally considered good, but there is a consensus that lower can be fine down to about 0.1. A summary and the total values are returned, with the summary being useful if several models are included in a list for fit argument
- **priorPredictive**: A plot of data simulated from the prior using plotPrior. This should generate data that is biologically plausible for your situation, but it will probably be much more variable than your data. That is the effect of the mildly informative thick tailed lognormal priors. If you specified non-default style priors then this currently will not work.

- **posteriorPredictive**: A plot of each model's posterior predictive interval over time. This is the same as plots returned from growthPlot and shows 1-99 coming to a mean yellow trend line. These should encompass the overwhelming majority of your data and ideally match the variance pattern that you see in your data. If parts of the predicted interval are biologically impossible (area below 0, percentage about 100 model should be reconsidered.

Value

A named list containing Rhat, ESS, NEFF, and Prior/Posterior Predictive plots. See details for interpretation.

See Also

plotPrior for visual prior predictive checks.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group, sigma = "logistic",
  df = simdf, start = list(
    "A" = 130, "B" = 12, "C" = 3,
    "sigmaA" = 20, "sigmaB" = 10, "sigmaC" = 2
  ), type = "brms"
)
fit_test <- fitGrowth(ss,
  iter = 600, cores = 1, chains = 1, backend = "cmdstanr",
  sample_prior = "only" # only sampling from prior for speed
)
barg(fit_test, ss)
fit_2 <- fit_test
fit_list <- list(fit_test, fit_2)
x <- barg(fit_list, list(ss, ss))
```

---

brmPlot *Function to visualize brms models similar to those made using growthSS outputs.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by `growthPlot`.

Usage

```
brmPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  hierarchy_value = NULL,
  vir_option = "plasma"
)
```

Arguments

fit	A brmsfit object, similar to those fit with growthSS outputs.
form	A formula similar to that in growthSS inputs specifying the outcome, predictor, and grouping structure of the data as outcome ~ predictor|individual/group.
df	An optional dataframe to use in plotting observed growth curves on top of the model.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
timeRange	An optional range of times to use. This can be used to view predictions for future data if the available data has not reached some point (such as asymptotic size), although prediction using splines outside of the observed range is not necessarily reliable.
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
hierarchy_value	If a hierarchical model is being plotted, what value should the hiearchical predictor be? If left NULL (the default) the mean value is used.
vir_option	Viridis color scale to use for plotting credible intervals. Defaults to "plasma".

Value

Returns a ggplot showing a brms model's credible intervals and optionally the individual growth lines.

Examples

```
simdf <- growthSim(
  "logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group, sigma = "spline",
  list("A" = 130, "B" = 10, "C" = 3),
  df = simdf, type = "brms"
```

```
)
fit <- fitGrowth(ss, backend = "cmdstanr", iter = 500, chains = 1, cores = 1)
growthPlot(fit = fit, form = y ~ time | group, groups = "a", df = ss$df)
```

`brmSurvPlot` *Function to visualize brms survival models specified using growthSS.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by `growthPlot`.

Usage

```
brmSurvPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE
)
```

Arguments

`fit`	A brmsfit object, similar to those fit with `growthSS` outputs.
`form`	A formula similar to that in `growthSS` inputs specifying the outcome, predictor, and grouping structure of the data as `outcome ~ predictor|individual/group`.
`df`	An optional dataframe to use in plotting observed growth curves on top of the model.
`groups`	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
`timeRange`	An optional range of times to use. This can be used to view predictions for future data if the available data has not reached some point (such as asymptotic size), although prediction using splines outside of the observed range is not necessarily reliable.
`facetGroups`	logical, should groups be separated in facets? Defaults to TRUE.

Value

Returns a ggplot showing a brms model's credible intervals and optionally the individual growth lines.

Examples

```
set.seed(123)
df <- growthSim("exponential",
  n = 20, t = 50,
  params = list("A" = c(1, 1), "B" = c(0.15, 0.1))
)
ss1 <- growthSS(
  model = "survival weibull", form = y > 100 ~ time | id / group,
  df = df, start = c(0, 5)
)
fit1 <- fitGrowth(ss1, iter = 600, cores = 2, chains = 2, backend = "cmdstanr")
brmSurvPlot(fit1, form = ss1$pcvrForm, df = ss1$df)

# note that using the cumulative hazard to calculate survival is likely to underestimate
# survival in these plots if events do not start immediately.
ss2 <- growthSS(
  model = "survival binomial", form = y > 100 ~ time | id / group,
  df = df, start = c(-4, 3)
)
fit2 <- fitGrowth(ss2, iter = 600, cores = 2, chains = 2, backend = "cmdstanr")
brmSurvPlot(fit2, form = ss2$pcvrForm, df = ss2$df)
```

`brmViolin` *Function to visualize hypotheses tested on brms models similar to those made using growthSS outputs.*

Description

Function to visualize hypotheses tested on brms models similar to those made using growthSS outputs.

Usage

```
brmViolin(fit, ss, hypothesis)
```

Arguments

`fit`	A brmsfit object or a dataframe of draws. If you need to combine multiple models then use combineDraws to merge their draws into a single dataframe for testing.
`ss`	A `pcvrss` object. The only component that is currently used is the pcvrForm.
`hypothesis`	A hypothesis expressed as a character string in the style of that used by `brms::hypothesis` and testGrowth. Note that currently this only supports hypotheses using two parameters from the model at a time (ie, "groupA / groupB > 1.05" works but "(groupA / groupB) - (groupC / groupD) > 1" does not). In the hypothesis "..." can be used to mean "all groups for this parameter" so that the hypothesis "... /

A_group1 > 1.05" would include all the "A" coefficients for groups 1:N along the x axis, see examples.

Value

Returns a ggplot showing a brms model's posterior distributions as violins and filled by posterior probability of some hypothesis.

Examples

```
set.seed(123)
simdf <- growthSim(
  "logistic",
  n = 20, t = 25,
 params = list("A" = c(200, 180, 190, 160), "B" = c(13, 11, 10, 10), "C" = c(3, 3, 3.25, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group, sigma = "int",
  list("A" = 130, "B" = 10, "C" = 3),
  df = simdf, type = "brms"
)

fit <- fitGrowth(ss, backend = "cmdstanr", iter = 500, chains = 1, cores = 1)
brmViolin(fit, ss, ".../A_groupd > 1.05") # all groups used
brmViolin(fit, ss, "A_groupa/A_groupd > 1.05") # only these two groups
```

---

bw.outliers *Remove outliers from bellwether data using cook's distance*

Description

Remove outliers from bellwether data using cook's distance

Usage

```
bw.outliers(
  df = NULL,
  phenotype,
  naTo0 = FALSE,
  group = c(),
  cutoff = 3,
  outlierMethod = "cooks",
  plotgroup = c("barcode", "rotation"),
  plot = TRUE,
  x = NULL,
  traitCol = "trait",
  valueCol = "value",
```

```
  labelCol = "label",
  idCol = NULL,
  ncp = NULL,
  separate = NULL
)
```

Arguments

df	Data frame to use. Can be in long or wide format.
phenotype	Column to use to classify outliers. If this is length > 1 then it is taken as the multi-value traits to use. See examples.
naTo0	Logical, should NA values to changed to 0.
group	Grouping variables to find outliers as a character vector. This is typically time and design variables (DAS, genotype, treatment, etc). These are used as predictors for ‘phenotype‘ in a generalized linear model.
cutoff	Cutoff for something being an "outlier" expressed as a multiplier on the mean of Cooks Distance for this data. This defaults to 5, with higher values being more conservative (removing less of the data).
outlierMethod	Method to be used in detecting outliers. Currently "cooks" and "mahalanobis" distances are supported, with "mahalanobis" only being supported for multi-value traits.
plotgroup	Grouping variables for drawing plots if plot=TRUE. Typically this is an identifier for images of a plant over time and defaults to c(’barcode’,"rotation").
plot	Logical, if TRUE then a list is returned with a ggplot and a dataframe.
x	Optional specification for x axis variable if plot is true. If left NULL (the default) then the first element of ‘group‘ is used.
traitCol	Column with phenotype names, defaults to "trait". This should generally not need to be changed from the default. If this and valueCol are present in colnames(df) then the data is assumed to be in long format.
valueCol	Column with phenotype values, defaults to "value". This should generally not need to be changed from the default.
labelCol	Column with phenotype labels for long data, defaults to "label". This should generally not need to be changed from the default.
idCol	Column(s) that identify individuals over time. Defaults to plotGroup.
ncp	Optionally specify the number of principle components to be used for MV data outlier detection with cooks distance. If left NULL (the default) then 3 will be used.
separate	Optionally separate the data by some variable to speed up the modeling step. If you have a design variable with very many levels then it may be helpful to separate by that variable. Note this will subset the data for each model so it will change the outlier removal (generally to be more conservative).

Value

The input dataframe with outliers removed and optionally a plot (if a plot is returned then output is a list).

Examples

```
sv <- growthSim("logistic",
  n = 5, t = 20,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
sv[130, ]$y <- 500
sv_res <- bw.outliers(
  df = sv, phenotype = "y", naTo0 = FALSE, cutoff = 15,
  group = c("time", "group"), outlierMethod = "cooks",
  plotgroup = "id", plot = TRUE
)
sv_res$plot

tryCatch(
  { # in case offline
    library(data.table)
    mvw <- read.pcv(paste0(
      "https://media.githubusercontent.com/media/joshqsumner/",
      "pcvrTestData/main/pcv4-multi-value-traits.csv"
    ), mode = "wide", reader = "fread")
    mvw$genotype <- substr(mvw$barcode, 3, 5)
    mvw$genotype <- ifelse(mvw$genotype == "002", "B73",
      ifelse(mvw$genotype == "003", "W605S",
        ifelse(mvw$genotype == "004", "MM", "Mo17")
      )
    )
    mvw$fertilizer <- substr(mvw$barcode, 8, 8)
    mvw$fertilizer <- ifelse(mvw$fertilizer == "A", "100",
      ifelse(mvw$fertilizer == "B", "50", "0")
    )
    mvw <- bw.time(mvw, timeCol = "timestamp", group = "barcode", plot = FALSE)

    phenotypes <- which(grepl("hue_freq", colnames(mvw)))

    mvw2 <- bw.outliers(
      df = mvw, phenotype = phenotypes, naTo0 = FALSE, outlierMethod = "cooks",
    group = c("DAS", "genotype", "fertilizer"), cutoff = 3, plotgroup = c("barcode", "rotation")
    )


    mvl <- read.pcv(paste0(
      "https://media.githubusercontent.com/media/joshqsumner/",
      "pcvrTestData/main/pcv4-multi-value-traits.csv"
    ), mode = "long")
    mvl$genotype <- substr(mvl$barcode, 3, 5)
    mvl$genotype <- ifelse(mvl$genotype == "002", "B73",
      ifelse(mvl$genotype == "003", "W605S",
        ifelse(mvl$genotype == "004", "MM", "Mo17")
      )
    )
    mvl$fertilizer <- substr(mvl$barcode, 8, 8)
```

```
    mvl$fertilizer <- ifelse(mvl$fertilizer == "A", "100",
      ifelse(mvl$fertilizer == "B", "50", "0")
    )
    mvl <- bw.time(mvl, timeCol = "timestamp", group = "barcode", plot = FALSE)

    mvl2 <- bw.outliers(
      df = mvl, phenotype = "hue_frequencies", naTo0 = FALSE, outlierMethod = "cooks",
    group = c("DAS", "genotype", "fertilizer"), cutoff = 3, plotgroup = c("barcode", "rotation")
    )
  },
  error = function(e) {
    message(e)
  }
)
```

---

bw.time *Time conversion and plotting for bellwether data*

Description

Time conversion and plotting for bellwether data

Usage

```
bw.time(
  df = NULL,
  mode = c("DAS", "DAP", "DAE"),
  plantingDelay = NULL,
  phenotype = NULL,
  cutoff = 1,
  timeCol = "timestamp",
  group = "Barcodes",
  plot = TRUE,
  format = "%Y-%m-%d %H:%M:%S",
  traitCol = "trait",
  valueCol = "value",
  index = NULL
)
```

Arguments

df	Data frame to use, this can be in wide or long format.
mode	One of "DAS", "DAP" or "DAE" (Days After Planting and Days After Emergence). Defaults to adding all columns. Note that if timeCol is not an integer then DAS is always returned.
plantingDelay	If ‘mode‘ includes "DAP" then ‘plantingDelay‘ is used to adjust "DAS"

phenotype	If ‘mode‘ includes "DAE" then this is the phenotype used to classify emergence.
cutoff	If ‘mode‘ inlcludes "DAE" then this value is used to classify emergence. Defaults to 1, meaning an image with a value of 1 or more for ‘phenotype‘ has "emerged".
timeCol	Column of input time values, defaults to "timestamp". If this is not an integer then it is assumed to be a timestamp in the format of the format argument.
group	Grouping variables to specify unique plants as a character vector. This defaults to "Barcodes". These taken together should identify a unique plant across time, although often "angle" or "rotation" should be added.
plot	Logical, should plots of the new time variables be printed?
format	An R POSIXct format, defaults to lemnatech standard format. This is only used if timeCol is not an integer.
traitCol	Column with phenotype names, defaults to "trait". This should generally not need to be changed from the default. If this and valueCol are present in colnames(df) then the data is assumed to be in long format.
valueCol	Column with phenotype values, defaults to "value". This should generally not need to be changed from the default.
index	Optionally a time to use as the beginning of the experiment. This may be useful if you have multiple datasets or you are adding data from bw.water and plants were watered before being imaged or if you want to index days off of midnight. This defaults to NULL but will take any value coercible to POSIXct by `as.POSIXct(... , tz="UTC")` such as "2020-01-01 18:30:00"

Value

The input dataframe with new integer columns for different ways of describing time in the experiment. If plot is TRUE then a ggplot is also returned as part of a list.

Examples

```
f <- "https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/pcv4-single-value-traits.csv"
tryCatch(
  {
    sv <- read.pcv(
      f,
      mode = "wide", reader = "fread"
    )
    sv$genotype = substr(sv$barcode, 3, 5)
    sv$genotype = ifelse(sv$genotype == "002", "B73",
      ifelse(sv$genotype == "003", "W605S",
        ifelse(sv$genotype == "004", "MM", "Mo17")
      )
    )
    sv$fertilizer = substr(sv$barcode, 8, 8)
    sv$fertilizer = ifelse(sv$fertilizer == "A", "100",
      ifelse(sv$fertilizer == "B", "50", "0")
    )
    sv <- bw.time(sv,
```

```
    plantingDelay = 0, phenotype = "area_pixels", cutoff = 10,
    timeCol = "timestamp", group = c("barcode", "rotation"), plot = FALSE
  )

  svl <- read.pcv(
    f,
    mode = "long", reader = "fread"
  )
  svl$genotype = substr(svl$barcode, 3, 5)
  svl$genotype = ifelse(svl$genotype == "002", "B73",
    ifelse(svl$genotype == "003", "W605S",
      ifelse(svl$genotype == "004", "MM", "Mo17")
    )
  )
  svl$fertilizer = substr(svl$barcode, 8, 8)
  svl$fertilizer = ifelse(svl$fertilizer == "A", "100",
    ifelse(svl$fertilizer == "B", "50", "0")
  )
  svl <- bw.time(svl,
    plantingDelay = 0, phenotype = "area_pixels", cutoff = 10, timeCol = "timestamp",
    group = c("barcode", "rotation"), plot = FALSE
  )
 },
 error = function(e) {
  message(e)
 }
)
```

`bw.water` *Read in lemnatech watering data from metadata.json files*

Description

Read in lemnatech watering data from metadata.json files

Usage

```
bw.water(file = NULL, envKey = "environment")
```

Arguments

`file`	Path to a json file of lemnatech metadata.
`envKey`	Character string representing the json key for environment data. By default this is set to "environment". Currently there are no situations where this makes sense to change.

Value

A data frame containing the bellwether watering data

Examples

```
tryCatch(
  {
   w <- bw.water("https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/metadata.json")
  },
  error = function(e) {
    message(e)
  }
)
```

---

checkGroups *Helper function to check groups in data.*

Description

Helper function to check groups in data.

Usage

```
checkGroups(df, group)
```

Arguments

df	Data frame to use.
group	Set of variables to use in grouping observations. These taken together should identify a unique plant (or unique plant at a unique angle) across time.

Value

If there are duplicates in the grouping then this will return a message with code to start checking the duplicates in your data.

Examples

```
df <- growthSim("linear",
  n = 10, t = 10,
  params = list("A" = c(2, 1.5))
)
checkGroups(df, c("time", "id", "group"))
df$time[12] <- 3
checkGroups(df, c("time", "id", "group"))
```

combineDraws *Combine Draws From brms Models*

Description

Helper function for binding draws from several brms models to make a data.frame for use with brms::hypothesis(). This will also check that the draws are comparable using basic model metrics.

Usage

```
combineDraws(..., message = TRUE)
```

Arguments

...	Some number of brmsfit objects and/or dataframes of draws (should generally be the same type of model fit to different data)
message	Logical, should messages about possible problems be printed? Default is TRUE. This will warn if models may not have converged, if there are different numbers of draws in the objects, or if models have different formulations.

Value

Returns a dataframe of posterior draws.

Examples

```
# note that this example will fit several bayesian models and may run for several minutes.

simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list(
    "A" = c(200, 160, 220, 200, 140, 300),
    "B" = c(13, 11, 10, 9, 16, 12),
    "C" = c(3, 3.5, 3.2, 2.8, 3.3, 2.5)
  )
)
ss_ab <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  sigma = "logistic", df = simdf[simdf$group %in% c("a", "b"), ],
  start = list(
    "A" = 130, "B" = 12, "C" = 3,
    "sigmaA" = 15, "sigmaB" = 10, "sigmaC" = 3
  ), type = "brms"
)

ss_cd <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  sigma = "logistic", df = simdf[simdf$group %in% c("c", "d"), ],
```

```
  start = list(
    "A" = 130, "B" = 12, "C" = 3,
    "sigmaA" = 15, "sigmaB" = 10, "sigmaC" = 3
  ), type = "brms"
)

ss_ef <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  sigma = "logistic", df = simdf[simdf$group %in% c("e", "f"), ],
  start = list(
    "A" = 130, "B" = 12, "C" = 3,
    "sigmaA" = 15, "sigmaB" = 10, "sigmaC" = 3
  ), type = "brms"
)
ss_ef2 <- growthSS(
  model = "gompertz", form = y ~ time | id / group,
  sigma = "logistic", df = simdf[simdf$group %in% c("e", "f"), ],
  start = list(
    "A" = 130, "B" = 12, "C" = 3,
    "sigmaA" = 15, "sigmaB" = 10, "sigmaC" = 3
  ), type = "brms"
)


fit_ab <- fitGrowth(ss_ab, chains = 1, cores = 1, iter = 1000)
fit_ab2 <- fitGrowth(ss_ab, chains = 1, cores = 1, iter = 1200)
fit_cd <- fitGrowth(ss_cd, chains = 1, cores = 1, iter = 1000)
fit_ef <- fitGrowth(ss_ef, chains = 1, cores = 1, iter = 1000)
fit_ef2 <- fitGrowth(ss_ef2, chains = 1, cores = 1, iter = 1000)

x <- combineDraws(fit_ab, fit_cd, fit_ef)
draws_ef <- as.data.frame(fit_ef)
draws_ef <- draws_ef[, grepl("^b_", colnames(draws_ef))]
x2 <- combineDraws(fit_ab2, fit_cd, draws_ef)
x3 <- combineDraws(fit_ab, fit_cd, fit_ef2)
```

`conjugate` *Bayesian testing using conjugate priors and method of moments for single or multi value traits.*

Description

Function to perform bayesian tests and ROPE comparisons using single or multi value traits with several distributions.

Usage

```
conjugate(
```

```
  s1 = NULL,
  s2 = NULL,
 method = c("t", "gaussian", "beta", "binomial", "lognormal", "lognormal2", "poisson",
   "negbin", "vonmises", "vonmises2", "uniform", "pareto", "gamma", "bernoulli",
  "exponential", "bivariate_uniform", "bivariate_gaussian", "bivariate_lognormal"),
  priors = NULL,
  plot = FALSE,
  rope_range = NULL,
  rope_ci = 0.89,
  cred.int.level = 0.89,
  hypothesis = "equal",
  support = NULL
)
```

Arguments

s1	A data.frame or matrix of multi value traits or a vector of single value traits. If a multi value trait is used then column names should include a number representing the "bin". Alternatively for distributions other than "binomial" (which requires list data with "successes" and "trials" as numeric vectors in the list, see examples) this can be a formula specifying `outcome ~ group` where group has exactly 2 levels. If using wide MV trait data then the formula should specify column positions ~ grouping such as `1:180 ~ group`. This sample is shown in red if plotted.
s2	An optional second sample, or if s1 is a formula then this should be a dataframe. This sample is shown in blue if plotted.
method	The distribution/method to use. Currently "t", "gaussian", "beta", "binomial", "lognormal", "lognormal2", "poisson", "negbin" (negative binomial), "uniform", "pareto", "gamma", "bernoulli", "exponential", "vonmises", and "vonmises2" are supported. The count (binomial, poisson and negative binomial), bernoulli, exponential, and pareto distributions are only implemented for single value traits due to their updating and/or the nature of the input data. The "t" and "gaussian" methods both use a T distribution with "t" testing for a difference of means and "gaussian" testing for a difference in the distributions (similar to a Z test). Both Von Mises options are for use with circular data (for instance hue values when the circular quality of the data is relevant). Note that non-circular distributions can be compared to each other. This should only be done with caution. Make sure you understand the interpretation of any comparison you are doing if you specify two methods (c("gaussian", "lognormal") as an arbitrary example). There are also 3 bivariate conjugate priors that are supported for use with single value data. Those are "bivariate_uniform", "bivariate_gaussian" and "bivariate_lognormal".
priors	Prior distributions described as a list of lists. If this is a single list then it will be duplicated for the second sample, which is generally a good idea if both samples use the same distribution (method). Elements in the inner lists should be named for the parameter they represent (see examples). These names vary by method (see details). By default this is NULL and weak priors (generally

	jeffrey's priors) are used. The `posterior` part of output can also be recycled as a new prior if Bayesian updating is appropriate for your use.
`plot`	Logical, should a ggplot be made and returned.
`rope_range`	Optional vector specifying a region of practical equivalence. This interval is considered practically equivalent to no effect. Kruschke (2018) suggests c(-0.1, 0.1) as a broadly reasonable ROPE for standardized parameters. That range could also be rescaled by a standard deviation/magnitude for non-standardized parameters, but ultimately this should be informed by your setting and scientific question. See Kruschke (2018) for details on ROPE and other Bayesian methods to aide decision-making doi:10.1177/2515245918771304 and doi:10.1037/a0029146.
`rope_ci`	The credible interval probability to use for ROPE. Defaults to 0.89.
`cred.int.level`	The credible interval probability to use in computing HDI for samples, defaults to 0.89.
`hypothesis`	Direction of a hypothesis if two samples are provided. Options are "unequal", "equal", "greater", and "lesser", read as "sample1 greater than sample2".
`support`	Optional support vector to include all possible values the random variable (samples) might take. This defaults to NULL in which case each method will use default behavior to attempt to calculate a dense support, but it is a good idea to supply this with some suitable vector. For example, the Beta method uses `seq(0.0001, 0.9999, 0.0001)` for support.

Details

Prior distributions default to be weakly informative and in some cases you may wish to change them.

- **"t" and "gaussian":** `priors = list( mu=c(0,0),n=c(1,1),s2=c(20,20) )` , where mu is the mean, n is the number of prior observations, and s2 is variance
- **"beta", "bernoulli", and "binomial":** `priors = list( a=c(0.5, 0.5), b=c(0.5, 0.5) )`, where a and b are shape parameters of the beta distribution. Note that for the binomial distribution this is used as the prior for success probability P, which is assumed to be beta distributed as in a beta-binomial distribution.
- **"lognormal":** `priors = list(mu = 0, sd = 5)` , where mu and sd describe the normal distribution of the mean parameter for lognormal data. Note that these values are on the log scale.
- **"lognormal2":** `priors = list(a = 1, b = 1)` , where a and b are the shape and scale parameters of the gamma distribution of lognormal data's precision parameter (using the alternative mu, precision paramterization).
- **"gamma":** `priors = list(shape = 0.5, scale = 0.5, known_shape = 1)`, where shape and scale are the respective parameters of the gamma distributed rate (inverse of scale) parameter of gamma distributed data.
- **"poisson" and "exponential":** `priors = list(a=c(0.5,0.5),b=c(0.5,0.5))`, where a and b are shape parameters of the gamma distribution.

- **"negbin":** `priors = list(r=c(10,10), a=c(0.5,0.5),b=c(0.5,0.5))`, where r is the r parameter of the negative binomial distribution (representing the number of successes required) and where a and b are shape parameters of the beta distribution. Note that the r value is not updated. The conjugate beta prior is only valid when r is fixed and known, which is a limitation for this method.
- **"uniform":** `list(scale = 0.5, location = 0.5)`, where scale is the scale parameter of the pareto distributed upper boundary and location is the location parameter of the pareto distributed upper boundary. Note that different sources will use different terminology for these parameters. These names were chosen for consistency with the `extraDistr` implementation of the pareto distribution. On Wikipedia the parameters are called shape and scale, corresponding to extraDistr's scale and location respecitvely, which can be confusing. Note that the lower boundary of the uniform is assumed to be 0.
- **"pareto":** `list(a = 1, b = 1, known_location = min(data))`, where a and b are the shape and scale parameters of the gamma distribution of the pareto distribution's scale parameter. In this case location is assumed to be constant and known, which is less of a limitation than knowing r for the negative binomial method since location will generally be right around/just under the minimum of the sample data. Note that the pareto method is only implemented currently for single value traits since one of the statistics needed to update the gamma distribution here is the product of the data and we do not currently have a method to calculate a similar sufficient statistic from multi value traits.
- **"vonmises":** `list(mu = 0, kappa = 0.5, boundary = c(-pi, pi), known_kappa = 1, n = 1)`, where mu is the direction of the circular distribution (the mean), kappa is the precision of the mean, boundary is a vector including the two values that are the where the circular data "wraps" around the circle, known_kappa is the fixed value of precision for the total distribution, and n is the number of prior observations. This Von Mises option updates the conjugate prior for the mean direction, which is itself Von-Mises distributed. This in some ways is analogous to the T method, but assuming a fixed variance when the mean is updated. Note that due to how the rescaling works larger circular boundaries can be slow to plot.
- **"vonmises2":** `priors = list(mu = 0, kappa = 0.5, boundary = c(-pi, pi), n = 1)`, where mu and kappa are mean direction and precision of the von mises distribution, boundary is a vector including the two values that are the where the circular data "wraps" around the circle, and n is the number of prior observations. This Von-Mises implementation does not assume constant variance and instead uses MLE to estimate kappa from the data and updates the kappa prior as a weighted average of the data and the prior. The mu parameter is then updated per Von-Mises conjugacy.
- **"bivariate_uniform":** `list(location_l = 1, location_u = 2, scale = 1)`, where scale is the shared scale parameter of the pareto distributed upper and lower boundaries and location l and u are the location parameters for the Lower (l) and Upper (u) boundaries of the uniform distribution. Note this uses the same terminology for the pareto distribution's parameters as the "uniform" method.
- **"bivariate_gaussian" and "bivariate_lognormal":** `list(mu = 0, sd = 10, a = 1, b = 1)`, where mu and sd are the mean and standard deviation of the Normal distribution of the data's mean and a and b are the shape and scale of the gamma distribution on precision. Note that internally this uses the Mu and Precision parameterization of the normal distribution and those are the parameters shown in the plot and tested, but priors use Mu and SD for the normal distribution of the mean.

See examples for plots of these prior distributions.

Value

A list with named elements:

- **summary**: A data frame containing HDI/HDE values for each sample and the ROPE as well as posterior probability of the hypothesis and ROPE test (if specified). The HDE is the "Highest Density Estimate" of the posterior, that is the tallest part of the probability density function. The HDI is the Highest Density Interval, which is an interval that contains X% of the posterior distribution, so `cred.int.level = 0.8` corresponds to an HDI that includes 80 percent of the posterior probability.
- **posterior**: A list of updated parameters in the same format as the prior for the given method. If desired this does allow for Bayesian updating.
- **plot_df**: A data frame of probabilities along the support for each sample. This is used for making the ggplot.
- **rope_df**: A data frame of draws from the ROPE posterior.
- **plot**: A ggplot showing the distribution of samples and optionally the distribution of differences/ROPE

Examples

```
mv_ln <- mvSim(
  dists = list(
    rlnorm = list(meanlog = log(130), sdlog = log(1.2)),
    rlnorm = list(meanlog = log(100), sdlog = log(1.3))
  ),
  n_samples = 30
)

# lognormal mv
ln_mv_ex <- conjugate(
  s1 = mv_ln[1:30, -1], s2 = mv_ln[31:60, -1], method = "lognormal",
  priors = list(mu = 5, sd = 2),
  plot = FALSE, rope_range = c(-40, 40), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal", support = NULL
)

# lognormal sv
ln_sv_ex <- conjugate(
  s1 = rlnorm(100, log(130), log(1.3)), s2 = rlnorm(100, log(100), log(1.6)),
  method = "lognormal",
  priors = list(mu = 5, sd = 2),
  plot = FALSE, rope_range = NULL, rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal", support = NULL
)

# Z test mv example

mv_gauss <- mvSim(
  dists = list(
    rnorm = list(mean = 50, sd = 10),
    rnorm = list(mean = 60, sd = 12)
```

```
  ),
  n_samples = 30
)

gauss_mv_ex <- conjugate(
  s1 = mv_gauss[1:30, -1], s2 = mv_gauss[31:60, -1], method = "gaussian",
  priors = list(mu = 30, n = 1, s2 = 100),
  plot = FALSE, rope_range = c(-25, 25), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal", support = NULL
)

# T test sv example

gaussianMeans_sv_ex <- conjugate(
  s1 = rnorm(10, 50, 10), s2 = rnorm(10, 60, 12), method = "t",
  priors = list(mu = 30, n = 1, s2 = 100),
  plot = FALSE, rope_range = c(-5, 8), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal", support = NULL
)

# beta mv example

set.seed(123)
mv_beta <- mvSim(
  dists = list(
    rbeta = list(shape1 = 5, shape2 = 8),
    rbeta = list(shape1 = 10, shape2 = 10)
  ),
  n_samples = c(30, 20)
)

beta_mv_ex <- conjugate(
  s1 = mv_beta[1:30, -1], s2 = mv_beta[31:50, -1], method = "beta",
  priors = list(a = 0.5, b = 0.5),
  plot = FALSE, rope_range = c(-0.1, 0.1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# beta sv example

beta_sv_ex <- conjugate(
  s1 = rbeta(20, 5, 5), s2 = rbeta(20, 8, 5), method = "beta",
  priors = list(a = 0.5, b = 0.5),
  plot = FALSE, rope_range = c(-0.1, 0.1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# binomial sv example
# note that specifying trials = 20 would also work
# and the number of trials will be recycled to the length of successes

binomial_sv_ex <- conjugate(
  s1 = list(successes = c(15, 14, 16, 11), trials = c(20, 20, 20, 20)),
```

```
  s2 = list(successes = c(7, 8, 10, 5), trials = c(20, 20, 20, 20)), method = "binomial",
  priors = list(a = 0.5, b = 0.5),
  plot = FALSE, rope_range = c(-0.1, 0.1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# poisson sv example

poisson_sv_ex <- conjugate(
  s1 = rpois(20, 10), s2 = rpois(20, 8), method = "poisson",
  priors = list(a = 0.5, b = 0.5),
  plot = FALSE, rope_range = c(-1, 1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# negative binomial sv example
# knowing r (required number of successes) is an important caveat for this method.
# in the current implementation we suggest using the poisson method for data such as leaf counts

negbin_sv_ex <- conjugate(
  s1 = rnbinom(20, 10, 0.5), s2 = rnbinom(20, 10, 0.25), method = "negbin",
  priors = list(r = 10, a = 0.5, b = 0.5),
  plot = FALSE, rope_range = c(-1, 1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# von mises mv example

mv_gauss <- mvSim(
  dists = list(
    rnorm = list(mean = 50, sd = 10),
    rnorm = list(mean = 60, sd = 12)
  ),
  n_samples = c(30, 40)
)
vm1_ex <- conjugate(
  s1 = mv_gauss[1:30, -1],
  s2 = mv_gauss[31:70, -1],
  method = "vonmises",
  priors = list(mu = 45, kappa = 1, boundary = c(0, 180), known_kappa = 1, n = 1),
  plot = FALSE, rope_range = c(-1, 1), rope_ci = 0.89,
  cred.int.level = 0.89, hypothesis = "equal"
)

# von mises 2 sv example
vm2_ex <- conjugate(
  s1 = brms::rvon_mises(10, 2, 2),
  s2 = brms::rvon_mises(15, 3, 3),
  method = "vonmises2",
  priors = list(mu = 0, kappa = 0.5, boundary = c(-pi, pi), n = 1),
  cred.int.level = 0.95,
  plot = FALSE
)
```

`cumulativePheno` *Reduce phenotypes in longitudinal data to cumulative sums of phenotypes.*

Description

Often in bellwether experiments we are curious about the effect of some treatment vs control. For certain routes in analysing the data this requires considering phenotypes as relative differences compared to a control.

Usage

```
cumulativePheno(
  df,
  phenotypes = NULL,
  group = "barcode",
  timeCol = "DAS",
  traitCol = "trait",
  valueCol = "value"
)
```

Arguments

`df`	Dataframe to use, this can be in long or wide format.
`phenotypes`	A character vector of column names for the phenotypes that should be compared against control.
`group`	A character vector of column names that identify groups in the data. Defaults to "barcode". These groups will be calibrated separately, with the exception of the group that identifies a control within the greater hierarchy.
`timeCol`	Column name to use for time data.
`traitCol`	Column with phenotype names, defaults to "trait". This should generally not need to be changed from the default. If this and valueCol are present in colnames(df) then the data is assumed to be in long format.
`valueCol`	Column with phenotype values, defaults to "value". This should generally not need to be changed from the default.

Value

A dataframe with cumulative sum columns added for specified phenotypes

Examples

```
f <- "https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/pcv4-single-value-traits.csv"
tryCatch(
  {
    sv <- read.pcv(
      f,
      reader = "fread"
    )
    sv$genotype <- substr(sv$barcode, 3, 5)
    sv$genotype <- ifelse(sv$genotype == "002", "B73",
      ifelse(sv$genotype == "003", "W605S",
        ifelse(sv$genotype == "004", "MM", "Mo17")
      )
    )
    sv$fertilizer <- substr(sv$barcode, 8, 8)
    sv$fertilizer <- ifelse(sv$fertilizer == "A", "100",
      ifelse(sv$fertilizer == "B", "50", "0")
    )

    sv <- bw.time(sv,
      plantingDelay = 0, phenotype = "area_pixels", cutoff = 10,
      timeCol = "timestamp", group = c("barcode", "rotation"), plot = TRUE
    )$data
    sv <- bw.outliers(sv,
      phenotype = "area_pixels", group = c("DAS", "genotype", "fertilizer"),
      plotgroup = c("barcode", "rotation")
    )$data
    phenotypes <- colnames(sv)[19:35]
    phenoForm <- paste0("cbind(", paste0(phenotypes, collapse = ", "), ")")
    groupForm <- "DAS+DAP+barcode+genotype+fertilizer"
    form <- as.formula(paste0(phenoForm, "~", groupForm))
    sv <- aggregate(form, data = sv, mean, na.rm = TRUE)
    pixels_per_cmsq <- 42.5^2 # pixel per cm^2
    sv$area_cm2 <- sv$area_pixels / pixels_per_cmsq
    sv$height_cm <- sv$height_pixels / 42.5
    df <- sv
    phenotypes <- c("area_cm2", "height_cm")
    group <- c("barcode")
    timeCol <- "DAS"
    df <- cumulativePheno(df, phenotypes, group, timeCol)


    sv_l <- read.pcv(
      f,
      mode = "long", reader = "fread"
    )
    sv_l$genotype <- substr(sv_l$barcode, 3, 5)
    sv_l$genotype <- ifelse(sv_l$genotype == "002", "B73",
      ifelse(sv_l$genotype == "003", "W605S",
        ifelse(sv_l$genotype == "004", "MM", "Mo17")
      )
    )
```

```
    sv_l$fertilizer <- substr(sv_l$barcode, 8, 8)
    sv_l$fertilizer <- ifelse(sv_l$fertilizer == "A", "100",
      ifelse(sv_l$fertilizer == "B", "50", "0")
    )
    sv_l <- bw.time(sv_l,
      plantingDelay = 0, phenotype = "area_pixels", cutoff = 10,
      timeCol = "timestamp", group = c("barcode", "rotation")
    )$data
    sv_l <- cumulativePheno(sv_l,
      phenotypes = c("area_pixels", "height_pixels"),
      group = c("barcode", "rotation"), timeCol = "DAS"
    )
  },
  error = function(e) {
    message(e)
  }
)
```

`distributionPlot` *Function for plotting iterations of posterior distributions*

Description

Function for plotting iterations of posterior distributions

Usage

```
distributionPlot(
  fits,
  form,
  df,
  priors = NULL,
  params = NULL,
  maxTime = NULL,
  patch = TRUE
)
```

Arguments

`fits`	A list of brmsfit objects following the same data over time. Currently checkpointing is not supported.
`form`	A formula describing the growth model similar to `growthSS` and `brmPlot` such as: outcome ~ predictor |individual/group
`df`	data used to fit models (this is used to plot each subject's trend line).
`priors`	a named list of samples from the prior distributions for each parameter in `params`. This is only used if sample_prior=FALSE in the brmsfit object. If left NULL then no prior is included.

params	a vector of parameters to include distribution plots of. Defaults to NULL which will use all parameters from the top level model.
maxTime	Optional parameter to designate a max time not observed in the models so far
patch	Logical, should a patchwork plot be returned or should lists of ggplots be returned?

Value

A ggplot or a list of ggplots (depending on patch).

Examples

```
f <- "https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/brmsFits.rdata"
tryCatch(
  {
    print(load(url(f)))
    library(brms)
    library(ggplot2)
    library(patchwork)
    fits <- list(fit_3, fit_15)
    form <- y~time | id / group
    priors <- list(
      "phi1" = rlnorm(2000, log(130), 0.25),
      "phi2" = rlnorm(2000, log(12), 0.25),
      "phi3" = rlnorm(2000, log(3), 0.25)
    )
    params <- c("A", "B", "C")
    d <- simdf
    maxTime <- NULL
    patch <- TRUE
    from3to25 <- list(
      fit_3, fit_5, fit_7, fit_9, fit_11,
      fit_13, fit_15, fit_17, fit_19, fit_21, fit_23, fit_25
    )
    distributionPlot(
      fits = from3to25, form = y ~ time | id / group,
      params = params, d = d, priors = priors, patch = FALSE
    )
    distributionPlot(
      fits = from3to25, form = y ~ time | id / group,
      params = params, d = d, patch = FALSE
    )
  },
  error = function(e) {
    message(e)
  }
)

## End(Not run)
```

`fitGrowth` *Ease of use wrapper function for fitting various growth models specified by growthSS*

Description

Ease of use wrapper function for fitting various growth models specified by growthSS

Usage

```
fitGrowth(ss, ...)
```

Arguments

`ss`	A list generated by `growthSS`.
`...`	Additional arguments passed to model fitting functions determined by `ss$type`. For type = "nlme" these are passed to `nlme::nlmeControl`, not `nlme::nlme`. Additional arguments are documented in fitGrowthbrms, fitGrowthnlme, fitGrowthnls, fitGrowthnlrq, fitGrowthmgcvgam, fitGrowthsurvreg, fitGrowthflexsurv.

Value

A fit model from the selected type.

See Also

growthPlot for model visualization, testGrowth for hypothesis testing, barg for Bayesian model reporting metrics.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | group,
  df = simdf, type = "nls"
)
fitGrowth(ss)
ss <- growthSS(
  model = "gam", form = y ~ time | group,
  df = simdf, type = "nls"
)
fitGrowth(ss)
```

`fitGrowthbrms` *Ease of use brms wrapper function for fitting growth models specified by* `growthSS`

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthbrms(
  ss,
  iter = 2000,
  cores = getOption("mc.cores", 1),
  chains = 4,
  prior = NULL,
  backend = "cmdstanr",
  silent = 0,
  ...
)

fitGrowthbrmsgam(
  ss,
  iter = 2000,
  cores = getOption("mc.cores", 1),
  chains = 4,
  prior = NULL,
  backend = "cmdstanr",
  silent = 0,
  ...
)
```

Arguments

`ss`	A list generated by `growthSS`.
`iter`	A number of iterations to sample for each chain. By default half this length is taken as warm-up for the MCMC algorithm. This defaults to 2000.
`cores`	A number of cores to run in parallel. This defaults to 1 if the "mc.cores" option is not set. Generally this is specified as one core per chain so that the model is fit in parallel.
`chains`	A number of markov chains to use, this defaults to 4.
`prior`	A `brmsprior` object if `growthSS` did not have priors specified. If left NULL (the default) and ss does not contain priors then a warning is issued but the model will still attempt to fit.
`backend`	A backend for brms to use Stan through. This defaults to use "cmdstanr".

silent	Passed to brms::brm to control verbosity. This defaults to 0, the most verbose option so that messages and progress are printed. With changes to cmdstanr and brms this may be removed, but the option will be available through .... Note that this is likely to print lots of messages during warmup iterations as the MCMC gets started.
...	Additional arguments passed to brms::brm.

Value

A brmsfit object, see ?`brmsfit-class` for details.

`fitGrowthflexsurv` *Ease of use wrapper function for fitting growth models specified by* `growthSS`

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthflexsurv(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to flexsurv::flexsurvreg.

Value

A survreg object.

`fitGrowthlm` *Ease of use lm wrapper function for fitting growth models specified by* `mvSS`

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthlm(ss, ...)
```

Arguments

ss	A list generated by mvSS.
...	Additional arguments passed to stats::lm.

Value

An lm object, see ?lm for details.

---

fitGrowthmgcvgam — *Ease of use mgcv wrapper function for fitting gams specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthmgcvgam(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to mgcv::gam.

Value

An gam object, see ?gam for details.

---

fitGrowthnlme — *Ease of use nlme wrapper function for fitting growth models specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnlme(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to nlme::nlmeControl.

Value

An nlme object, see ?nlme for details.

`fitGrowthnlmegam` *Ease of use lme wrapper function for fitting gams specified by* `growthSS`

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnlmegam(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to nlme::lmeControl.

Value

An lme object, see ?lme for details.

`fitGrowthnlrq` *Ease of use nlrq wrapper function for fitting growth models specified by* `growthSS`

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnlrq(ss, cores = getOption("mc.cores", 1), ...)
```

Arguments

ss	A list generated by growthSS.
cores	Optionally specify how many cores to run in parallel if ss$taus is >1L. Defaults to 1 if mc.cores option is not set globally.
...	Additional arguments passed to quantreg::nlrq.

Value

An nlrqModel object if tau is length of 1 or a list of such models for multiple taus, see ?quantreg::nlrq or ?nls::nlsModel for details.

fitGrowthnlrqgam — *Ease of use rq wrapper function for fitting gams specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnlrqgam(ss, cores = getOption("mc.cores", 1), ...)
```

Arguments

ss	A list generated by growthSS.
cores	number of cores to run in parallel
...	Additional arguments passed to quantreg::rq.

Value

An rq object, see ?rq for details.

fitGrowthnls — *Ease of use nls wrapper function for fitting growth models specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnls(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to stats::nls.

Value

An nls object, see ?nls for details.

fitGrowthnlsgam *Ease of use lm wrapper function for fitting gams specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthnlsgam(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to stats::lm.

Value

An lm object, see ?lm for details.

fitGrowthrq *Ease of use rq wrapper function for fitting models specified by* mvSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthrq(ss, cores = getOption("mc.cores", 1), ...)
```

Arguments

ss	A list generated by mvSS.
cores	number of cores to run in parallel
...	Additional arguments passed to quantreg::rq.

Value

An rq object, see ?rq for details.

fitGrowthsurvreg — *Ease of use wrapper function for fitting growth models specified by* growthSS

Description

Helper function generally called from fitGrowth.

Usage

```
fitGrowthsurvreg(ss, ...)
```

Arguments

ss	A list generated by growthSS.
...	Additional arguments passed to survival::survreg.

Value

A survreg object.

flexsurvregPlot — *Function to visualize* flexsurv::flexsurvreg *models fit by* fitGrowth.

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by growthPlot.

Usage

```
flexsurvregPlot(
  fit,
  form,
  groups = NULL,
  df = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

fit	A model fit returned by fitGrowth with type="nls".
form	A formula similar to that in growthSS inputs (or the pcvrForm part of the output) specifying the outcome, predictor, and grouping structure of the data as outcome ~ predictor|individual/group. If the individual and group are specified then the observed growth lines are plotted.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
df	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for nls models.
timeRange	Ignored, included for compatibility with other plotting functions.
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an survival model's survival function.

Examples

```
df <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "survival weibull", form = y > 100 ~ time | id / group,
  df = df, type = "flexsurv"
)
fit <- fitGrowth(ss)
flexsurvregPlot(fit, form = ss$pcvrForm, df = ss$df, groups = "a")
flexsurvregPlot(fit,
  form = ss$pcvrForm, df = ss$df,
  facetGroups = FALSE, groupFill = TRUE
)
```

---

frem *Variance partitioning using Full Random Effects Models*

Description

Variance partitioning for phenotypes (over time) using fully random effects models

Usage

```
frem(
  df,
  des,
  phenotypes,
  timeCol = NULL,
  cor = TRUE,
  returnData = FALSE,
  combine = TRUE,
  markSingular = FALSE,
  time = NULL,
  time_format = "%Y-%m-%d",
  ...
)
```

Arguments

df	Dataframe containing phenotypes and design variables, optionally over time.
des	Design variables to partition variance for as a character vector.
phenotypes	Phenotype column names (data is assumed to be in wide format) as a character vector.
timeCol	A column of the data that denotes time for longitudinal experiments. If left NULL (the default) then all data is assumed to be from one timepoint.
cor	Logical, should a correlation plot be made? Defaults to TRUE.
returnData	Logical, should the used to make plots be returned? Defaults to FALSE.
combine	Logical, should plots be combined with patchwork? Defaults to TRUE, which works well when there is a single timepoint being used.
markSingular	Logical, should singular fits be marked in the variance explained plot? This is FALSE by default but it is good practice to check with TRUE in some situations. If TRUE this will add white markings to the plot where models had singular fits, which is the most common problem with this type of model.
time	If the data contains multiple timepoints then which should be used? This can be left NULL which will use the maximum time if timeCol is specified. If a single number is provided then that time value will be used. Multiple numbers will include those timepoints. The string "all" will include all timepoints.
time_format	Format for non-integer time, passed to strptime, defaults to "%Y-%m-%d".
...	Additional arguments passed to lme4::lmer.

Value

Returns either a plot (if returnData=FALSE) or a list with a plot and data/a list of dataframes (depending on returnData and cor).

Examples

```
library(data.table)
set.seed(456)
df <- data.frame(
  genotype = rep(c("g1", "g2"), each = 10),
  treatment = rep(c("C", "T"), times = 10),
  time = rep(c(1:5), times = 2),
  date_time = rep(paste0("2024-08-", 21:25), times = 2),
  pheno1 = rnorm(20, 10, 1),
  pheno2 = sort(rnorm(20, 5, 1)),
  pheno3 = sort(runif(20))
)
out <- frem(df, des = "genotype", phenotypes = c("pheno1", "pheno2", "pheno3"), returnData = TRUE)
lapply(out, class)
frem(df,
  des = c("genotype", "treatment"), phenotypes = c("pheno1", "pheno2", "pheno3"),
  cor = FALSE
)
frem(df,
  des = "genotype", phenotypes = c("pheno1", "pheno2", "pheno3"),
  combine = FALSE, timeCol = "time", time = "all"
)
frem(df,
  des = "genotype", phenotypes = c("pheno1", "pheno2", "pheno3"),
  combine = TRUE, timeCol = "time", time = 1
)
frem(df,
  des = "genotype", phenotypes = c("pheno1", "pheno2", "pheno3"),
  cor = FALSE, timeCol = "time", time = 3:5, markSingular = TRUE
)
df[df$time == 3, "genotype"] <- "g1"
frem(df,
  des = "genotype", phenotypes = c("pheno1", "pheno2", "pheno3"),
  cor = FALSE, timeCol = "date_time", time = "all", markSingular = TRUE
)
```

`gam_diff` *Helper function for visualizing differences in GAMs fit with* `mgcv::gam`

Description

Note that using GAMs will be less useful than fitting parameterized models as supported by `growthSS` and `fitGrowth` for common applications in plant phenotyping.

Usage

```
gam_diff(
  model,
  newdata = NULL,
  g1,
  g2,
  byVar = NULL,
  smoothVar = NULL,
  cis = seq(0.05, 0.95, 0.05),
  unconditional = TRUE,
  plot = TRUE
)
```

Arguments

model	A model fit with smooth terms by mgcv::gam
newdata	A data.frame of new data to use to make predictions. If this is left NULL (the default) then an attempt is made to make newdata using the first smooth term in the formula. See examples for guidance on making appropriate newdata
g1	A character string for the level of byVar to use as the first group to compare, if plot=TRUE then this will be shown in blue.
g2	The second group to compare (comparison will be g1 - g2). If plot=TRUE then this will be shown in red.
byVar	Categorical variable name used to separate splines as a string.
smoothVar	The variable that splines were used on. This will often be a time variable.
cis	Confidence interval levels, can be multiple. For example, 0.95 would return Q_0.025 and Q_0.975 columns, and c(0.9, 0.95) would return Q_0.025, Q_0.05, Q_0.95, and Q_0.975 columns. Defaults to seq(0.05, 0.95, 0.05)
unconditional	Logical, should unconditional variance-covariance be used in calculating standard errors. Defaults to TRUE.
plot	Logical, should a plot of the difference be returned? Defaults to TRUE.

Value

A dataframe or a list containing a ggplot and a dataframe

Examples

```
ex <- pcvr::growthSim("logistic",
  n = 20, t = 25,
  params = list(
    "A" = c(200, 160),
    "B" = c(13, 11),
    "C" = c(3, 3.5)
  )
)
```

```
m <- mgcv::gam(y ~ group + s(time, by = factor(group)), data = ex)

support <- expand.grid(
  time = seq(min(ex$time), max(ex$time), length = 400),
  group = factor(unique(ex$group))
)

out <- gam_diff(
  model = m, newdata = support, g1 = "a", g2 = "b",
  byVar = "group", smoothVar = "time", plot = TRUE
)
dim(out$data)
out$plot
out2 <- gam_diff(
  model = m, g1 = "a", g2 = "b", byVar = NULL, smoothVar = NULL, plot = TRUE
)
```

`growthPlot` *Function to visualize models made by fitGrowth.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function.

Usage

```
growthPlot(
  fit,
  form,
  groups = NULL,
  df = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = !facetGroups,
  hierarchy_value = NULL
)
```

Arguments

`fit`	A model fit object (or a list of `nlrq` models) as returned by `fitGrowth`.
`form`	A formula similar to that in `growthSS` inputs (or the `pcvrForm` part of the output) specifying the outcome, predictor, and grouping structure of the data as `outcome ~ predictor|individual/group`. Generally this is given directly from the growthSS output (`ss$pcvrForm`). If the formula does not include both individuals and groups then lines from the data will not be plotted which may be best if your data does not specify unique individuals and your model does not include autocorrelation.

groups — An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.

df — A dataframe to use in plotting observed growth curves on top of the model and for making predictions.

timeRange — An optional range of times to use. This can be used to view predictions for future data if the avaiable data has not reached some point (such as asymptotic size).

facetGroups — logical, should groups be separated in facets? Defaults to TRUE.

groupFill — logical, should groups have different colors? Defaults to the opposite of facetGroups. If TRUE then viridis colormaps are used in the order c('plasma', 'mako', 'viridis', 'inferno', 'cividis', 'magma', 'turbo', 'rocket'). Alternatively this can be given as a vector of viridis colormap names to use in a different order than above. Note that for brms models this is ignored except if used to specify a different viridis color map to use.

hierarchy_value — If a hierarchical model is being plotted, what value should the hiearchical predictor be? If left NULL (the default) the mean value is used.

Value

Returns a ggplot showing a brms model's credible intervals and optionally the individual growth lines.

See Also

growthSS and fitGrowth for making compatible models, testGrowth for hypothesis testing on compatible models.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  df = simdf, type = "nls"
)
fit <- fitGrowth(ss)
growthPlot(fit, form = ss$pcvrForm, df = ss$df)
```

growthSim *Growth data simulating function*

Description

growthSim can be used to help pick reasonable parameters for common growth models to use in prior distributions or to simulate data for example models/plots.

Usage

```
growthSim(
  model = c("logistic", "gompertz", "double logistic", "double gompertz",
    "monomolecular", "exponential", "linear", "power law", "frechet", "weibull",
    "gumbel", "logarithmic", "bragg", "lorentz", "beta"),
  n = 20,
  t = 25,
  params = list(),
  D = 0
)
```

Arguments

model	One of "logistic", "gompertz", "weibull", "frechet", "gumbel", "monomolecular", "exponential", "linear", "power law", "logarithmic", "bragg", "double logistic", or "double gompertz". Alternatively this can be a pseudo formula to generate data from a segmented growth curve by specifying "model1 + model2", see examples and growthSS. Decay can be specified by including "decay" as part of the model such as "logistic decay" or "linear + linear decay". Count data can be specified with the "count: " prefix, similar to using "poisson: model" in growthSS. While "gam" models are supported by growthSS they are not simulated by this function.
n	Number of individuals to simulate over time per each group in params
t	Max time (assumed to start at 1) to simulate growth to as an integer.
params	A list of numeric parameters. A, B, C notation is used in the order that parameters appear in the formula (see examples). Number of groups is inferred from the length of these vectors of parameters. In the case of the "double" models there are also A2, B2, and C2 terms. Changepoints should be specified as "changePointX" or "fixedChangePointX" as in growthSS.
D	If decay is being simulated then this is the starting point for decay. This defaults to 0.

Details

The params argument requires some understanding of how each growth model is parameterized. Examples of each are below should help, as will the examples.

- **Logistic**: ‘A / (1 + exp((B-x)/C))‘ Where A is the asymptote, B is the inflection point, C is the growth rate.
- **Gompertz**: ‘A * exp(-B * exp(-C*x))‘ Where A is the asymptote, B is the inflection point, C is the growth rate.
- **Weibull**: ‘A * (1-exp(-(x/C)^B))‘ Where A is the asymptote, B is the weibull shape parameter, C is the weibull scale parameter.
- **Frechet**: ‘A * exp(-((x-0)/C)^(-B))‘ Where A is the asymptote, B is the frechet shape parameter, C is the frechet scale parameter. Note that the location parameter (conventionally m) is 0 in these models for simplicity but is still included in the formula.
- **Gumbel**: ‘A * exp(-exp(-(x-B)/C))‘ Where A is the asymptote, B is the inflection point (location), C is the growth rate (scale).
- **Double Logistic**: ‘A / (1+exp((B-x)/C)) + ((A2-A) /(1+exp((B2-x)/C2)))‘ Where A is the asymptote, B is the inflection point, C is the growth rate, A2 is the second asymptote, B2 is the second inflection point, and C2 is the second growth rate.
- **Double Gompertz**: ‘A * exp(-B * exp(-C*x)) + ((A2-A) * exp(-B2 * exp(-C2*(x-B))))‘ Where A is the asymptote, B is the inflection point, C is the growth rate, A2 is the second asymptote, B2 is the second inflection point, and C2 is the second growth rate.
- **Monomolecular**: ‘A-A * exp(-B * x)‘ Where A is the asymptote and B is the growth rate.
- **Exponential**: ‘A * exp(B * x)‘ Where A is the scale parameter and B is the growth rate.
- **Linear**: ‘A * x‘ Where A is the growth rate.
- **Logarithmic**: ‘A * log(x)‘ Where A is the growth rate.
- **Power Law**: ‘A * x ^ (B)‘ Where A is the scale parameter and B is the growth rate.
- **Bragg**: ‘A * exp(-B * (x - C) ^ 2)‘ This models minima and maxima as a dose-response curve where A is the max response, B is the "precision" or slope at inflection, and C is the x position of the max response.
- **Lorentz**: ‘A / (1 + B * (x - C) ^ 2)‘ This models minima and maxima as a dose-response curve where A is the max response, B is the "precision" or slope at inflection, and C is the x position of the max response. Generally Bragg is preferred to Lorentz for dose-response curves.
- **Beta**: ‘A * (((x - D) / (C - D)) * ((E - x) / (E - C)) ^ ((E - C) / (C - D))) ^ B‘ This models minima and maxima as a dose-response curve where A is the Maximum Value, B is a shape/concavity exponent similar to the sum of alpha and beta in a Beta distribution, C is the position of maximum value, D is the minimum position where distribution > 0, E is the maximum position where distribution > 0. This is a difficult model to fit but can model non-symmetric dose-response relationships which may sometimes be worth the extra effort.

Note that for these distributions parameters generally do not exist in a vacuum. Changing one will make the others look different in the resulting data. The examples are a good place to start if you are unsure what parameters to use.

Value

Returns a dataframe of example growth data following the input parameters.

Examples

```
library(ggplot2)
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Logistic")

simdf <- growthSim("gompertz",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(0.2, 0.25))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Gompertz")

simdf <- growthSim("weibull",
  n = 20, t = 25,
  params = list("A" = c(100, 100), "B" = c(1, 0.75), "C" = c(2, 3))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "weibull")

simdf <- growthSim("frechet",
  n = 20, t = 25,
  params = list("A" = c(100, 110), "B" = c(2, 1.5), "C" = c(5, 2))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "frechet")

simdf <- growthSim("gumbel",
  n = 20, t = 25,
  list("A" = c(120, 140), "B" = c(6, 5), "C" = c(4, 3))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "gumbel")

simdf <- growthSim("double logistic",
  n = 20, t = 70,
  params = list(
    "A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5),
    "A2" = c(400, 300), "B2" = c(35, 40), "C2" = c(3.25, 2.75)
  )
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Double Logistic")
```

```
simdf <- growthSim("double gompertz",
  n = 20, t = 100,
  params = list(
    "A" = c(180, 140), "B" = c(13, 11), "C" = c(0.2, 0.2),
    "A2" = c(400, 300), "B2" = c(50, 50), "C2" = c(0.1, 0.1)
  )
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Double Gompertz")

simdf <- growthSim("monomolecular",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(0.08, 0.1))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Monomolecular")

simdf <- growthSim("exponential",
  n = 20, t = 25,
  params = list("A" = c(15, 20), "B" = c(0.095, 0.095))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Exponential")

simdf <- growthSim("linear",
  n = 20, t = 25,
  params = list("A" = c(1.1, 0.95))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Linear")

simdf <- growthSim("logarithmic",
  n = 20, t = 25,
  params = list("A" = c(2, 1.7))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Logarithmic")

simdf <- growthSim("power law",
  n = 20, t = 25,
  params = list("A" = c(16, 11), "B" = c(0.75, 0.7))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "Power Law")

simdf <- growthSim("bragg",
```

```
  n = 20, t = 100,
  list("A" = c(10, 15), "B" = c(0.01, 0.02), "C" = c(50, 60))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "bragg")

# simulating models from segmented growth models

simdf <- growthSim(
  model = "linear + linear", n = 20, t = 25,
 params = list("linear1A" = c(16, 11), "linear2A" = c(0.75, 0.7), "changePoint1" = c(11, 14))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "linear + linear")

simdf <- growthSim(
  model = "linear + linear decay", n = 20, t = 25,
  params = list("linear1A" = c(16, 11), "linear2A" = c(3, 2), "changePoint1" = c(11, 14))
)
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "linear + linear decay")

simdf <- growthSim(
  model = "linear + linear + logistic", n = 20, t = 50,
  params = list(
    "linear1A" = c(16, 11), "linear2A" = c(3, 4), # linear slopes, very intuitive
    "changePoint1" = c(11, 14), "changePoint2" = c(10, 12),
    # changepoint1 is standard, changepoint2 happens relative to changepoint 1
    "logistic3A" = c(200, 210), "logistic3B" = c(20, 25), "logistic3C" = c(3, 3)
  )
)
# similar to changepoint2, the asymptote and inflection point are relative to the starting
# point of the logistic growth component. This is different than the model output
# if you were to fit a curve to this model using `growthSS`.
ggplot(simdf, aes(time, y, group = interaction(group, id))) +
  geom_line(aes(color = group)) +
  labs(title = "linear + linear + logistic")
```

`growthSS` *Ease of use growth model helper function.*

Description

Output from this should be passed to fitGrowth to fit the specified model.

Usage

```
growthSS(
  model,
  form,
  sigma = NULL,
  df,
  start = NULL,
  pars = NULL,
  type = "brms",
  tau = 0.5,
  hierarchy = NULL
)
```

Arguments

model	The name of a model as a character string. Supported options are c("logistic", "gompertz", "weibull", "frechet", "gumbel", "monomolecular", "exponential", "linear", "power law", "bragg", "lorentz", "beta", "double logistic", "double gompertz", "gam", "int"), with "int" representing an intercept only model which is only used in brms (and is expected to only be used in threshold models or to model homoskedasticity). Note that the dose response curves (bragg, lorentz, and beta) may be difficult to fit using the `nlme` backend but should work well using other options. See `growthSim` for examples of each type of single parameterized growth curve ("gam" is not supported in `growthSim`). You can also specify decay models by including the "decay" keyword with the model name. Note that using "decay" is only necessary for the brms backend since otherwise the priors are strictly positive. In brms models the entire formula is negated for decay models so that lognormal priors can still be used when at least some coefficients would be negative. Additionally, the "int_" prefix may be added to a model name to specify that an intercept should be included. By default these models are assumed to have intercepts at 0, which is often fine. If you include an intercept in a brms model then you would specify the prior as you would for an "A", "B", or "C" parameter but as "I". By default growthSS will make student T priors for intercept parameters in the same way that it will for estimated changepoints (see below). With type="brms" you can also specify segmented models by combining model names with a plus sign such as "linear + linear". In a segmented model the names for parameters do not follow the normal "A", "B", "C" notation, instead they are named for the type of model, the position in the formula, then for the parameter of that model. There will also be parameters to represent the time when growth switches from one model to another called either "changepointX" or "fixedChangePointX". All "changePointX" terms are estimated as parameters of the model. "fixedChangePointX" parameters are not estimated and are kept as the numeric value given in the priors, this is useful if your experiment has an intervention at a set time which you expect to change the growth process acutely. For the "linear + linear" example this would yield parameters "linear1A", "changePoint1" (or "fixedChangePoint1"), and "linear2A". A "linear + gompertz" model would have "linear1A", "changePoint1", "gompertz2A", "gompertz2B", and "gompertz2C" for parameters. Note that double

sigmoid models are not supported as parts of segmented models and gams can currently only be included as the last part of a segmented model. When using a changepoint model it may be worth using segments that are simpler to fit (gompertz instead of EVD options, for instance). Currently "homo" and "int" are treated the same and "spline" and "gam" are interchangeable. Time-to-event models can be specified using the "survival" keyword, see details for an explanation of the changes that entails. Similarly, using the brms backend response distributions (see `brms::brmsfamily`) can be specified in the model as "family: model" so that a model of logistic increasing counts may be written as `model = "poisson: logistic"`.

`form` A formula describing the model. The left hand side should only be the outcome variable (phenotype), and a cutoff if you are making a survival model (see details). The right hand side needs at least the x variable (typically time). Grouping is also described in this formula using roughly lme4 style syntax,with formulas like `y~time|individual/group` to show that predictors should vary by `group` and autocorrelation between `individual:group` interactions should be modeled. Note that autocorrelation is only modeled with the "brms" backend in this way. "nlme" requires random effects and correlations to use the same grouping, so autocorrelation using the "nlme" backend works at the group level, so will slightly underestimate the autocorrelation at the individual level. If group has only one level or is not included then it will be ignored in formulas for growth and variance (this may be the case if you split data before fitting models to be able to run more smaller models each more quickly). To include multiple grouping variables they should be separated with "+" as in `y~time|individual/groupingVariable1 + groupingVariable2`. For some backends multiple grouping variables will be combined into a single factor of their interaction. Hierarchical models can be specified for the brms backend as `y~time+other_covariate|individual/group` in which case the parameters of the main growth model will themselves be estimated by models as specified in the `hierarchy` argument. For instance, if normally "A" had an intercept for each `group`, now it would be predicted as `A ~ AI + AA * covariate` where AI and AA now have an intercept for each `group`. Note that if you specify a hierarchical model then priors are required for AI and AA in the previous example.

`sigma` Other models for distributional parameters. This argument is only used with "brms" and "nlme" models and is handled differently for each. When type="brms" this can be supplied as a model or as a list of models. It is turned into a formula (or list of formulas) with an entry corresponding to each distributional parameter (after the mean) of the growth model family. If no family was specified (`model="logistic"` for instance) then the student T distribution is used, with additional distributional parameters sigma and nu. To check the naming of distributional parameters in each response family use `brms::brmsfamily("family")$dpars`. The supported options are the same as the model options (including threshold models). For distributional parameters that do not have a formula specified they will be modeled as intercept only (not by group). Parameter names are the same as those in the main model but with the distributional parameter name as a prefix. Additionally, if a linear model is used for sigma then it can be modeled with or without a prior, if a prior is specified ("sigmaA") then a non-linear formula is used and the "sigmaA" parameter will be included in the output instead of the de-

fault "sigma" term. In the rare case that you wish to model the mean and the 3rd distributional parameter but not the 2nd then `sigma = list("not_estimated", "model")` would allow for that. When type ="nlme" the options are more limited to c("none", "power", "exp"), corresponding to using `nlme::varIdent`, `nlme::varPower`, or `nlme::varExp` respectively where "power" is the default.

`df` A dataframe to use. Must contain all the variables listed in the formula. Note that rows with NA or infinite values in x, y, or hierarchical predictors are removed.

`start` An optional named list of starting values OR means for prior distributions. If this is not provided then starting values are picked with `stats::selfStart`. When type = "brms" these should be provided and are treated as the means of lognormal priors for all growth model parameters and T_5(mu, 3) priors for changepoint parameters. This is done because the values are strictly positive and the lognormal distribution is easily interpreted. The changepoint priors are T distributions for symmetry, 5 DF having been chosen for heavy but not unmanageable tails. If this argument is not provided then priors are made using brms::get_prior. Those priors are unlikely to be suitable and a different set of priors will need to be made for the model using `brms::set_prior` for good convergence. When specifying starting values/prior means think of this as being similar to the `params` argument in `growthSim`. Names should correspond to parameter names from the `model` argument. A numeric vector can also be used, but specifying names is best practice for clarity. Additionally, due to a limitation in `brms` currently lower bounds cannot be set for priors for specific groups. If priors include multiple groups (`start = list(A = c(10,15), ...)`) then you will see warnings after the model is fit about not having specified a lower bound explicitly. Those warnings can safely be ignored and will be addressed if the necessary features are added to `brms`. See details for guidance.

`pars` Optionally specify which parameters should change by group. Not this is model dependent and is not implemented for brms models due to their more flexible hypothesis testing.

`type` Type of model to fit, options are "brms", "nlrq", "nlme", "nls", and "mgcv". Note that the "mgcv" option only supports "gam" models. Survival models can use the "survreg" model type (this will be called if any non-brms/flexsurv type is given) or the "flexsurv" model type which requires the flexsurv package to be installed. Note that for non-brms models variables in the model will be labeled by the factor level of the group, not necessarily by the group name. This is done for ease of use with different modeling functions, the levels are alphabetically sorted and can be checked using: `table(ss$df$group, ss$df$group_numericLabel)`.

`tau` A vector of quantiles to fit for nlrq models.

`hierarchy` Optionally a list of model parameters that should themselves by modeled by another predictor variable. This is only used with the brms backend.

Details

Default priors are not provided, but these can serve as starting points for each distribution. You are encouraged to use `growthSim` to consider what kind of trendlines result from changes to your prior and for interpretation of each parameter. The plotPrior function can be used to do prior predictive checks. You should not looking back and forth at your data trying to match your observed growth

exactly with a prior distribution, rather this should be informed by an understanding of the plants you are using and expectations based on previous research. For the "double" models the parameter interpretation is the same as for their non-double counterparts except that there are A and A2, etc. It is strongly recommended to familiarize yourself with the double sigmoid distributions using growthSim before attempting to model one. Additionally, those distributions are intended for use with long delays in an experiment, think stress recovery experiments, not for minor hiccups in plant growth.

- **Logistic**: `list('A' = 130, 'B' = 12, 'C' = 3)`
- **Gompertz**: `list('A' = 130, 'B' = 12, 'C' = 1.25)`
- **Weibull**: `list('A' = 130, 'B' = 2, 'C' = 2)`
- **Frechet**: `list('A' = 130, 'B' = 5, 'C' = 6)`
- **Gumbel**: `list('A' = 130, 'B' = 6, 'C' = 4)`
- **Double Logistic**: `list('A' = 130, 'B' = 12, 'C' = 3, 'A2' = 200, 'B2' = 25, 'C2' = 1)`
- **Double Gompertz**: `list('A' = 130, 'B' = 12, 'C' = 0.25, 'A2' = 220, 'B2' = 30, 'C2' = 0.1)`
- **Monomolecular**: `list('A' = 130, 'B' = 2)`
- **Exponential**: `list('A' = 15, 'B' = 0.1)`
- **Linear**: `list('A' = 1)`
- **Power Law**: `list('A' = 13, 'B' = 2)`

See details below about parameterization for each model option.

- **Logistic**: ‘A / (1 + exp((B-x)/C))‘ Where A is the asymptote, B is the inflection point, C is the growth rate.
- **Gompertz**: ‘A * exp(-B * exp(-C*x))‘ Where A is the asymptote, B is the inflection point, C is the growth rate.
- **Weibull**: ‘A * (1-exp(-(x/C)^B))‘ Where A is the asymptote, B is the weibull shape parameter, C is the weibull scale parameter.
- **Frechet**: ‘A * exp(-((x-0)/C)^(-B))‘ Where A is the asymptote, B is the frechet shape parameter, C is the frechet scale parameter. Note that the location parameter (conventionally m) is 0 in these models for simplicity but is still included in the formula.
- **Gumbel**: ‘A * exp(-exp(-(x-B)/C))‘ Where A is the asymptote, B is the inflection point (location), C is the growth rate (scale).
- **Double Logistic**: ‘A / (1+exp((B-x)/C)) + ((A2-A) /(1+exp((B2-x)/C2)))‘ Where A is the asymptote, B is the inflection point, C is the growth rate, A2 is the second asymptote, B2 is the second inflection point, and C2 is the second growth rate.
- **Double Gompertz**: ‘A * exp(-B * exp(-C*x)) + ((A2-A) * exp(-B2 * exp(-C2*(x-B))))‘ Where A is the asymptote, B is the inflection point, C is the growth rate, A2 is the second asymptote, B2 is the second inflection point, and C2 is the second growth rate.
- **Monomolecular**: ‘A-A * exp(-B * x)‘ Where A is the asymptote and B is the growth rate.
- **Exponential**: ‘A * exp(B * x)‘ Where A is the scale parameter and B is the growth rate.
- **Linear**: ‘A * x‘ Where A is the growth rate.

- **Power Law**: 'A * x^(B)' Where A is the scale parameter and B is the growth rate.
- **Bragg**: 'A * exp(-B * (x - C) ^ 2)' This models minima and maxima as a dose-response curve where A is the max response, B is the "precision" or slope at inflection, and C is the x position of the max response.
- **Lorentz**: 'A / (1 + B * (x - C) ^ 2)' This models minima and maxima as a dose-response curve where A is the max response, B is the "precision" or slope at inflection, and C is the x position of the max response. Generally Bragg is preferred to Lorentz for dose-response curves.
- **Beta**: 'A * (((x - D) / (C - D)) * ((E - x) / (E - C)) ^ ((E - C) / (C - D))) ^ B' This models minima and maxima as a dose-response curve where A is the Maximum Value, B is a shape/concavity exponent similar to the sum of alpha and beta in a Beta distribution, C is the position of maximum value, D is the minimum position where distribution > 0, E is the maximum position where distribution > 0. This is a difficult model to fit but can model non-symmetric dose-response relationships which may sometimes be worth the extra effort.

Note that for these distributions parameters do not exist in a vacuum. Changing one will make the others look different in the resulting data. The `growthSim` function can be helpful in familiarizing further with these distributions.

Using the `brms` backend the `sigma` argument optionally specifies a sub model to account for heteroskedasticity. Any combination of models (except for decay models) can be specified in the `sigma` term. If you need variance to raise and lower then a gam/spline is the most appropriate option.

Using the `brms` backend a model with lots of parameters may be difficult to estimate if there are lots of groups. If you have very many levels of your "group" variable in a complex model then consider fitting models to subsets of the "group" variable and using combineDraws to make a data.frame for hypothesis testing.

Limits on the Y variable can be specified in the `brms` backend. This should generally be unnecessary and will make the model slower to fit and potentially more difficult to set priors on. If you do have a limited phenotype (besides the normal positive constraint for growth models) then this may be helpful, one situation may be canopy coverage percentage which is naturally bounded at an upper and lower limit. To specify these limits add square brackets to the Y term with upper and lower limits such as `"y[0,100] ~ time|id/group"`. Other "Additional response information" such as resp_weights or standard errors can be specified using the `brms` backend, with those options documented fully in the `brms::brmsformula` details.

There are also three supported submodel options for `nlme` models, but a `varFunc` object can also be supplied, see `?nlme::varClasses`.

- **none**: `varIdent(1|group)`, which models a constant variance separately for each group.
- **power**: `varPower(x|group)`, which models variance as a power of x per group.
- **exp**: `varExp(x|group)`, which models variance as an exponent of x per group.

Survival models can be fit using the "survival" keyword in the model specification. Using the "brms" backend (type argument) you can specify "weibull" (the default) or "binomial" for the distribution to use in that model so that the final model string would be "survival binomial" or "survival weibull" which is equivalent to "survival". Time to event data is very different than standard phenotype data, so the formula argument should include a cutoff for the Y variable to count as an "event". For example, if you were checking germination using area and wanted to use 50 pixels as a germinated plant your formula would be `area > 50 ~ time|id/group`. Internally the input

dataframe will be converted to time-to-event data based on that formula. Alternatively you can make your own time to event data and supply that to growthSS. In that case your data should have columns called "n_events" (number of individuals experiencing the event at this time) and "n_eligible" (number of individuals who had not experienced the event at least up to this time) for the binomial model family OR "event" (binary 1,0 for TRUE, FALSE) for the Weibull model family. Note that since these are linear models using different model families the priors are handled differently. For survival models the default priors are weak regularizing priors (Normal(0,5)) on all parameters. If you wish to specify your own priors you can supply them as brmsprior objects or as a list such as `priors = list("group1" = c(0,3), "group2" = c(0,1))` where the order of values is Mu, Sigma. Any non-brms backend will instead use `survival::survreg` to fit the model unless the "flexsurv" type is specified. Distributions will be passed to `survreg` where options are "weibull", "exponential", "gaussian", "logistic","lognormal" and "loglogistic" if type = "survreg" or to `flexsurv::flexsurvreg` if type = "flexsurv" where options are "gengamma", "gengamma.orig", "genf", "genf.orig", "weibull", "gamma", "exp", "llogis", "lnorm", "gompertz", "exponential", and "lognormal". In `flexsurvreg` distributional modeling is supported and additional formula can be passed as a list to the sigma argument of growthSS in the same way as to the anc argument of `flexsurv::flexsurvreg`. Further additional arguments should be supplied via `fitGrowth` if desired.

Value

A named list of elements to make it easier to fit non linear growth models with several R packages.

For `brms` models the output contains:

`formula`: A `brms::bf` formula specifying the growth model, autocorrelation, variance submodel, and models for each variable in the growth model. `prior`: A brmsprior/data.frame object. `initfun`: A function to randomly initialize chains using a random draw from a gamma distribution (confines initial values to positive and makes correct number of initial values for chains and groups). `df` The data input, with dummy variables added if needed and a column to link groups to their factor levels. `family` The model family, currently this will always be "student". `pcvrForm` The form argument unchanged. This is returned so that it can be used later on in model visualization. Often it may be a good idea to save the output of this function with the fit model, so having this can be useful later on.

For `quantreg::nlrq` models the output contains:

`formula`: An `nls` style formula specifying the growth model with groups if specified. `taus`: The quantiles to be fit `start`: The starting values, typically these will be generated from the growth model and your data in a similar way as shown in `stats::selfStart` models. `df` The input data for the model. `pcvrForm` The form argument unchanged.

For `nls` models the output is the same as for `quantreg::nlrq` models but without `taus` returned.

For `nlme::nlme` models the output contains:

`formula`: An list of `nlme` style formulas specifying the model, fixed and random effects, random effect grouping, and variance model (weights). `start`: The starting values, typically these will be generated from the growth model and your data in a similar way as shown in `stats::selfStart` models. `df` The input data for the model. `pcvrForm` The form argument unchanged.

For all models the type and model are also returned for simplicity downstream.

See Also

fitGrowth for fitting the model specified by this list and mvSS for the multi-value trait equivalent.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  sigma = "spline", df = simdf,
  start = list("A" = 130, "B" = 12, "C" = 3), type = "brms"
)
lapply(ss, class)
ss$initfun()
# the next step would typically be compiling/fitting the model
# here we use very few chains and very few iterations for speed, but more of both is better.

fit_test <- fitGrowth(ss,
  iter = 500, cores = 1, chains = 1, backend = "cmdstanr",
  control = list(adapt_delta = 0.999, max_treedepth = 20)
)



# formulas and priors will look different if there is only one group in the data

ex <- growthSim("linear", n = 20, t = 25, params = list("A" = 2))
ex_ss <- growthSS(
  model = "linear", form = y ~ time | id / group, sigma = "spline",
  df = ex, start = list("A" = 1), type = "brms"
)

ex_ss$prior # no coef level grouping for priors
ex_ss$formula # intercept only model for A

ex2 <- growthSim("linear", n = 20, t = 25, params = list("A" = c(2, 2.5)))
ex2_ss <- growthSS(
  model = "linear", form = y ~ time | id / group, sigma = "spline",
  df = ex2, start = list("A" = 1), type = "brms"
)
ex2_ss$prior # has coef level grouping for priors
ex2_ss$formula # specifies an A intercept for each group and splines by group for sigma
```

---

mvSim *Multi Value Trait simulating function*

Description

mvSim can be used to simulate data for example models/plots.

Usage

```
mvSim(
  dists = list(rnorm = list(mean = 100, sd = 15)),
  n_samples = 10,
  counts = 1000,
  min_bin = 1,
  max_bin = 180,
  wide = TRUE,
  binwidth = 1,
  t = NULL,
  model = "linear",
  params = list(A = 10)
)
```

Arguments

dists	A list of lists, with names corresponding to random deviate generating functions and arguments to the function in the list values (see examples). Note that the n argument does not need to be provided.
n_samples	Number of samples per distribution to generate. Defaults to 10, can be >1L.
counts	Number of counts per histogram, defaults to 1000.
min_bin	The minumum bin number. This can be thought of as the minimum value that will be accepted in the distribution functions, with lower numbers being raised to this value. Note that bin arguments are both ignored in the case of "rbeta" and treated as 0,1.
max_bin	The number of bins to return. Note that this is also the max value that will be accepted in the distribution functions, with higher numbers being shrunk to this value. Defaults to 180.
wide	Boolean, should data be returned in wide format (the default)? If FALSE then long data is returned.
binwidth	How wide should bins be? Defaults to 1.
t	Number of timepoints to simulate. Defaults to NULL in which case data is simulated as non-longitudinal. Note that currently the first non n argument of the data simulating function is assumed to be the parameter changing over time (ie, mean in rnorm, meanlog in rlnorm).
model	A type of growth model, passed to growthSim. This is only used if t is specified.
params	Parameters for the growth model, passed to growthSim. This is also only used if t is specified. Note growth will start from the values specified in dists. See examples.

Value

Returns a dataframe of example multi-value trait data simulated from specified distributions.

Examples

```
library(extraDistr) # for rmixnorm
library(ggplot2)
dists <- list(
  rmixnorm = list(mean = c(70, 150), sd = c(15, 5), alpha = c(0.3, 0.7)),
  rnorm = list(mean = 90, sd = 3)
)
x <- mvSim(dists = dists, wide = FALSE)
dim(x)
x2 <- mvSim(dists = dists)
dim(x2)

ggplot(x, aes(
  x = as.numeric(sub("sim_", "", variable)),
  y = value, group = interaction(group, id), fill = group
)) +
  geom_col(position = "identity", alpha = 0.25) +
  pcv_theme() +
  labs(x = "bin")
dists = list(rnorm = list(mean = 30, sd = 15), rnorm = list(mean = 25, sd = 10))
x3 <- mvSim(
  dists = dists, wide = FALSE, # here we make longitudinal data
  t = 10, model = "linear", params = list("A" = c(10, 5))
)
ggplot(x3, aes(
  x = as.numeric(sub("sim_", "", variable)),
  y = value, group = interaction(group, id), fill = group
)) +
  facet_wrap(~times) +
  geom_col(position = "identity", alpha = 0.25) +
  pcv_theme() +
  labs(x = "bin")
```

---

mvSS *Ease of use multi-value trait model helper function.*

Description

This function provides a simplified interface to modeling multi-value traits using growthSS. Output from this should be passed to fitGrowth to fit the specified model.

Usage

```
mvSS(
  model = "linear",
  form,
  sigma = NULL,
  df,
```

```
  start = NULL,
  pars = NULL,
  type = "brms",
  tau = 0.5,
  hierarchy = NULL,
 spectral_index = c("none", "ari", "ci_rededge", "cri550", "cri700", "egi", "evi",
   "gdvi", "mari", "mcari", "mtci", "ndre", "ndvi", "pri", "psnd_chlorophyll_a",
   "psnd_chlorophyll_b", "psnd_caroteniods", "psri", "pssr_chlorophyll_a",
  "pssr_chlorophyll_b", "pssr_caroteniods", "rgri", "rvsi", "savi", "sipi", "sr",
   "vari", "vi_green", "wi", "fvfm", "fqfm")
)
```

Arguments

`model`	A model specification as in growthSS.
`form`	A formula similar to `label | value ~ time + id/group` where label is a column of histogram bins, value is the counts within those bins, time is an optional time variable, id identifies an individual, and group contains the treatment groups. If the time variable is not included then the individual variable should also not be included.
`sigma`	Distributional models passed to growthSS.
`df`	Data passed to growthSS.
`start`	Starting values or priors, passed to growthSS.
`pars`	Parameters to vary, passed to growthSS.
`type`	Backend to use, passed to growthSS.
`tau`	Quantile to model, passed to growthSS.
`hierarchy`	Formulae describing any hierarchical models, see growthSS.
`spectral_index`	Optionally, a spectral index from those calculated by PlantCV. If this is given then the appropriate truncation and model family (if applicable) will be included for the index you are using without you having to write it in the formula.

Value

A named list of plots showing prior distributions that `growthSS` would use, optionally with a plot of simulated growth curves using draws from those priors.

See Also

fitGrowth for fitting the model specified by this list.

Examples

```
set.seed(123)
mv_df <- mvSim(dists = list(rnorm = list(mean = 100, sd = 30)), wide = FALSE)
mv_df$group <- rep(c("a", "b"), times = 900)
mv_df <- mv_df[mv_df$value > 0, ]
mv_df$label <- as.numeric(gsub("sim_", "", mv_df$variable))
```

```
ss1 <- mvSS(
  model = "linear", form = label | value ~ group, df = mv_df,
  start = list("A" = 5), type = "brms", spectral_index = "none"
)

mod1 <- fitGrowth(ss1, backend = "cmdstanr", iter = 1000, chains = 1, cores = 1)
growthPlot(mod1, ss1$pcvrForm, df = ss1$df)


# when the model is longitudinal the same model is possible with growthSS

m1 <- mvSim(
  dists = list(
    rnorm = list(mean = 100, sd = 30),
    rnorm = list(mean = 110, sd = 25),
    rnorm = list(mean = 120, sd = 20),
    rnorm = list(mean = 135, sd = 15)
  ),
  wide = FALSE, n = 6
)
m1$time <- rep(1:4, times = 6 * 180)
m2 <- mvSim(
  dists = list(
    rnorm = list(mean = 85, sd = 25),
    rnorm = list(mean = 95, sd = 20),
    rnorm = list(mean = 105, sd = 15),
    rnorm = list(mean = 110, sd = 15)
  ),
  wide = FALSE, n = 6
)
m2$time <- rep(1:4, times = 6 * 180)
mv_df2 <- rbind(m1, m2)
mv_df2$group <- rep(c("a", "b"), each = 4320)
mv_df2 <- mv_df2[mv_df2$value > 0, ]
mv_df2$label <- as.numeric(gsub("sim_", "", mv_df2$variable))
ss_mv0 <- mvSS(
  model = "linear", form = label | value ~ group, df = mv_df2,
  start = list("A" = 50), type = "brms", spectral_index = "ci_rededge"
)
ss_mv0 # non longitudinal model setup

ss_mv1 <- mvSS(
  model = "linear", form = label | value ~ time | group, df = mv_df2,
  start = list("A" = 50), type = "brms", spectral_index = "ci_rededge"
)
ss_mv1
ss_mv2 <- growthSS(
  model = "skew_normal: linear",
  form = label | resp_weights(value) + trunc(lb = -1, ub = Inf) ~ time | group,
  df = mv_df2, start = list("A" = 50)
)
ss_mv2
```

```
# ignoring environments and other such details these are identical except for the
# function call.
unlist(lapply(names(ss_mv1), function(nm) {
  if (!identical(ss_mv1[[nm]], ss_mv2[[nm]],
    ignore.environment = TRUE,
    ignore.srcref = TRUE
  )) {
    if (!identical(as.character(ss_mv1[[nm]]), as.character(ss_mv2[[nm]]))) {
      nm
    }
  }
}))


if (rlang::is_installed("mnormt")) {
  m2 <- fitGrowth(ss_mv1, backend = "cmdstanr", iter = 1000, chains = 1, cores = 1)
  growthPlot(m2, ss_mv1$pcvrForm, df = ss_mv1$df)
}
```

---

mv_ag *Multi Value Trait Aggregation function*

Description

EMD can get very heavy with large datasets. For an example lemnatech dataset filtering for images from every 5th day there are 6332^2 = 40,094,224 pairwise EMD values. In long format that's a 40 million row dataframe, which is unwieldy. This function is to help reduce the size of datasets before comparing histograms and moving on with matrix methods or network analysis.

Usage

```
mv_ag(
  df,
  group,
  mvCols = "frequencies",
  n_per_group = 1,
  outRows = NULL,
  keep = NULL,
  parallel = getOption("mc.cores", 1),
  traitCol = "trait",
  labelCol = "label",
  valueCol = "value",
  id = "image"
)
```

Arguments

df	A dataframe with multi value traits. This can be in wide or long format, data is assumed to be long if traitCol, valueCol, and labelCol are present.
group	Vector of column names for variables which uniquely identify groups in the data to summarize data over. Typically this would be the design variables and a time variable.
mvCols	Either a vector of column names/positions representing multi value traits or a character string that identifies the multi value trait columns as a regex pattern. Defaults to "frequencies".
n_per_group	Number of rows to return for each group.
outRows	Optionally this is a different way to specify how many rows to return. This will often not be exact so that groups have the same number of observations each.
keep	A vector of single value traits to also average over groups, if there are a mix of single and multi value traits in your data.
parallel	Optionally the groups can be run in parallel with this number of cores, defaults to 1 if the "mc.cores" option is not set globally.
traitCol	Column with phenotype names, defaults to "trait".
labelCol	Column with phenotype labels (units), defaults to "label".
valueCol	Column with phenotype values, defaults to "value".
id	Column that uniquely identifies images if the data is in long format. This is ignored when data is in wide format.

Value

Returns a dataframe summarized by the specified groups over the multi-value traits.

Examples

```
s1 <- mvSim(
  dists = list(runif = list(min = 15, max = 150)),
  n_samples = 10,
  counts = 1000,
  min_bin = 1,
  max_bin = 180,
  wide = TRUE
)
mv_ag(s1, group = "group", mvCols = "sim_", n_per_group = 2)
```

net.plot *Visualizing igraph networks*

Description

Easy igraph visualization with pcv.net output

Usage

```
net.plot(
  net,
  fill = "strength",
  shape = NULL,
  size = 3,
  edgeWeight = "emd",
  edgeFilter = NULL
)
```

Arguments

net	Network object similar to that returned from pcv.net, having dataframes named "edges" and "nodes"
fill	Variable name(s) from the nodes data to be used to color points. By default "strength" is used.
shape	Optional discrete variable name(s) from the nodes data to be used to change the shape of points. If this variable is numeric it will be coerced to character.
size	Size of points, defaults to 3.
edgeWeight	Edge dataframe column to weight connections between nodes. Defaults to "emd" for compatability with pcv.emd.
edgeFilter	How should edges be filtered? This can be either a numeric (0.5) in which case it is taken as a filter where only edges with values greater than or equal to that number are kept or a character string ("0.5") in which case the strongest X percentage of edges are kept. This defaults to NULL which does no filtering, although that should not be considered the best standard behaviour. See details.

Value

Returns a ggplot of a network.

Examples

```
library(extraDistr)
dists <- list(
  rmixnorm = list(mean = c(70, 150), sd = c(15, 5), alpha = c(0.3, 0.7)),
  rnorm = list(mean = 90, sd = 3)
)
```

```
x <- mvSim(
  dists = dists, n_samples = 5, counts = 1000,
  min_bin = 1, max_bin = 180, wide = TRUE
)
emd_df <- pcv.emd(x,
  cols = "sim", reorder = c("group"), mat = FALSE,
  plot = FALSE, parallel = 1
)
net <- pcv.net(emd_df, meta = "group")
net.plot(net)
net.plot(net, edgeFilter = "0.25")
net.plot(net,
  edgeFilter = 0.25, fill = c("degree", "group"),
  shape = c("degree", "group")
)
net.plot(net,
  edgeFilter = 0.25, fill = c("degree", "group"),
  shape = c("degree")
)
```

`nlmePlot` *Function to visualize common* `nlme::nlme` *growth models.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by `growthPlot`.

Usage

```
nlmePlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

`fit`	A model fit returned by `fitGrowth` with type="nlme".
`form`	A formula similar to that in `growthSS` inputs (or the `pcvrForm` part of the output) specifying the outcome, predictor, and grouping structure of the data as `outcome ~ predictor|individual/group`

df	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for nlme models.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
timeRange	An optional range of times to use. This can be used to view predictions for future data if the avaiable data has not reached some point (such as asymptotic size).
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps.
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an nlme model's credible intervals and optionally the individual growth lines.

Examples

```
simdf <- growthSim("logistic",
  n = 10, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)

ss <- growthSS(
  model = "logistic", form = y ~ time | id / group, sigma = "none",
  df = simdf, start = NULL, type = "nlme"
)

fit <- fitGrowth(ss)

nlmePlot(fit, form = ss$pcvrForm, groups = NULL, df = ss$df, timeRange = NULL)
nlmePlot(fit, form = ss$pcvrForm, groups = "a", df = ss$df, timeRange = 1:10, groupFill = TRUE)
```

---

nlrqPlot *Function to visualize common* quantreg::nlrq *growth models.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by growthPlot.

Usage

```
nlrqPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

fit	A model fit, or list of model fits, returned by fitGrowth with type="nlrq".
form	A formula similar to that in growthSS inputs (or the pcvrForm part of the output) specifying the outcome, predictor, and grouping structure of the data as outcome ~ predictor|individual/group. If the individual and group are specified then the observed growth lines are plotted.
df	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for nlrq models.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
timeRange	An optional range of times to use. This can be used to view predictions for future data if the avaiable data has not reached some point (such as asymptotic size).
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an nlrq model's quantiles and optionally the individual growth lines.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  tau = c(0.5, 0.9), df = simdf, start = NULL, type = "nlrq"
)
fit <- fitGrowth(ss)
```

```
nlrqPlot(fit, form = ss$pcvrForm, df = ss$df, groups = "a", timeRange = 1:20)
nlrqPlot(fit, form = ss$pcvrForm, df = ss$df, groupFill = TRUE, virMaps = c("plasma", "viridis"))

ss <- growthSS(
  model = "logistic", form = y ~ time,
  tau = c(0.5, 0.9), df = simdf, start = NULL, type = "nlrq"
)
fit <- fitGrowth(ss)
nlrqPlot(fit, form = ss$pcvrForm, df = ss$df)
```

`nlsPlot` *Function to visualize common* `stats::nls` *growth models.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by `growthPlot`.

Usage

```
nlsPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)

gamPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)

lmPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
```

```
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

fit	A model fit returned by fitGrowth with type="nls".
form	A formula similar to that in growthSS inputs (or the pcvrForm part of the output) specifying the outcome, predictor, and grouping structure of the data as outcome ~ predictor|individual/group. If the individual and group are specified then the observed growth lines are plotted.
df	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for nls models.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
timeRange	An optional range of times to use. This can be used to view predictions for future data if the avaiable data has not reached some point (such as asymptotic size).
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an nls model's predictions.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "logistic", form = y ~ time | id / group,
  df = simdf, start = NULL, type = "nls"
)
fit <- fitGrowth(ss)
nlsPlot(fit, form = ss$pcvrForm, df = ss$df, groupFill = TRUE)
nlsPlot(fit, form = ss$pcvrForm, df = ss$df, groups = "a", timeRange = 1:10)

simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
```

```
)
ss <- growthSS(
  model = "gam", form = y ~ time | id / group,
  df = simdf, start = NULL, type = "nls"
)
fit <- fitGrowth(ss)
gamPlot(fit, form = ss$pcvrForm, df = ss$df, groupFill = TRUE)
gamPlot(fit, form = ss$pcvrForm, df = ss$df, groups = "a", timeRange = 1:10)
ss <- growthSS(
  model = "gam", form = y ~ time | group,
  df = simdf, start = NULL, type = "nls"
)
fit <- fitGrowth(ss)
gamPlot(fit, form = ss$pcvrForm, df = ss$df, groupFill = TRUE)

simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "gam", form = y ~ time | id / group,
  df = simdf, start = NULL, type = "nls"
)
fit <- fitGrowth(ss)
lmPlot(fit, form = ss$pcvrForm, df = ss$df)
```

---

pcadf *Function to run a PCA, plot and optionally return the data with PCA coordinates and pca object*

Description

Function to run a PCA, plot and optionally return the data with PCA coordinates and pca object

Usage

```
pcadf(
  df = NULL,
  cols = NULL,
  color = NULL,
  facet = NULL,
  returnData = TRUE,
  ncp = NULL
)
```

Arguments

df	Dataframe to ordinate

cols	columns to reduce dimensions of. Can be specified with names or positions. If this is length of 1 then it is treated as regex pattern to match the column names that should be used.
color	column name(s) used to color points in the pca plot.
facet	Optional column or vector to facet plots on.
returnData	Logical, should data be returned?
ncp	Optional, number of principal components to return attached to dataframe if data is returned. Defaults to all.

Details

If data is returned then it will contain the coordinates from the PCA and will not contain the columns that were reduced.

Value

A ggplot or list with a ggplot, a dataframe with the data and PCs, and the factominer PCA object as elements.

Examples

```
dists <- list(
  rlnorm = list(meanlog = log(40), sdlog = 0.5),
  rnorm = list(mean = 60, sd = 10)
)
mv <- mvSim(
  dists = dists, n_samples = 100, counts = 1000,
  min_bin = 1, max_bin = 180, wide = TRUE
)
mv$otherGroup <- sample(c("a", "b"), size = nrow(mv), replace = TRUE)
pcadf(mv, cols = "sim_", returnData = TRUE)
pcadf(mv, cols = 2:181, color = c("group", "otherGroup"), returnData = FALSE)
```

---

pcv.emd *Earth Mover's Distance between spectral histograms*

Description

pcv.emd can be used to calculate Earth Mover's Distance between pairwise histograms in a wide dataframe of multi value traits. The is expected to be used with output from mv_ag. See also pcv.euc for euclidean distance between histograms.

Usage

```
pcv.emd(
  df,
  cols = NULL,
  reorder = NULL,
  include = reorder,
  mat = FALSE,
  plot = TRUE,
  parallel = getOption("mc.cores", 1),
  trait = "trait",
  id = "image",
  value = "value",
  raiseError = TRUE,
  method = "emd"
)

pcv.euc(
  df,
  cols = NULL,
  reorder = NULL,
  include = reorder,
  mat = FALSE,
  plot = TRUE,
  parallel = getOption("mc.cores", 1),
  trait = "trait",
  id = "image",
  value = "value",
  raiseError = TRUE,
  method = "euc"
)
```

Arguments

df	Data frame to use with multi value traits in wide format or long format
cols	Columns to use. Defaults to NULL in which case all columns are used. Single strings will be used to regex a pattern in column names (see examples). A vector of names, positions, or booleans will also work. For long data this is taken as a regex pattern (or full name) to use in filtering the trait column.
reorder	Should data be reordered to put similar rows together in the resulting plot? This takes a vector of column names of length 1 or more (see examples).
include	if a long dataframe is returned then these columns will be added to the dataframe, labelled for i and j (the row positions for compared histograms). If a matrix is returned then this information is stored in the row names. This defaults to reorder.
mat	Logical, should data be returned as an nrow x nrow matrix or as a long dataframe? By Default this is FALSE and a long dataframe is returned. Both options are

	comparable in terms of speed, although for large datasets the matrix version may be slightly faster.
plot	Logical, should a plot be returned? For a matrix this is made with heatmap(), for a dataframe this uses ggplot.
parallel	Number of cores to use. Defaults to 1 unless the "mc.cores" option is set.
trait	Column name for long data to identify traits. This defaults to "trait". If this and value are in the column names of the data then it is assumed to be in long format, otherwise it is assumed to be in wide format.
id	A vector of column names that uniquely identifies observations if the data is in long format. Defaults to "image".
value	A column name for the values to be drawn from in long data. Defaults to "value".
raiseError	Logical, should warnings/errors be raised for potentially large output? It is easy to ask for very many comparisons with this function so the goal of this argument is to catch a few of those and give estimates of how much time something may take. If the function is expected to take very long then a warning or an error is raised. If this is set to FALSE then no time estimates are made.
method	Which method to use (one of "emd" or "euc"). Defaults to "emd".

Value

A dataframe/matrix (if plot=FALSE) or a list with a dataframe/matrix and\ a ggplot (if plot=TRUE). The returned data contains pairwise EMD values.

Examples

```
set.seed(123)
test <- mvSim(
  dists = list(
    runif = list(min = 0, max = 100),
    rnorm = list(mean = 90, sd = 20)
  ),
  n_samples = 10
)
test$meta1 <- rep(LETTERS[1:3], length.out = nrow(test))
test$meta2 <- rep(LETTERS[4:5], length.out = nrow(test))

x <- pcv.emd(
  df = test, cols = "sim", reorder = "group",
  include = c("meta1", "meta2"), mat = FALSE,
  plot = FALSE, parallel = 1
)
head(x)
x2 <- pcv.emd(
  df = test, cols = "sim", reorder = "group",
  include = c("meta1", "meta2"), mat = FALSE,
  plot = FALSE, parallel = 1, method = "euc"
)
head(x2)
```

```
tryCatch(
  {
    library(data.table)
    file <- paste0(
      "https://media.githubusercontent.com/media/joshqsumner/",
      "pcvrTestData/main/pcv4-multi-value-traits.csv"
    )
    df1 <- read.pcv(file, "wide", reader = "fread")

    df1$genotype <- substr(df1$barcode, 3, 5)
    df1$genotype <- ifelse(df1$genotype == "002", "B73",
      ifelse(df1$genotype == "003", "W605S",
        ifelse(df1$genotype == "004", "MM", "Mo17")
      )
    )
    df1$fertilizer <- substr(df1$barcode, 8, 8)
    df1$fertilizer <- ifelse(df1$fertilizer == "A", "100",
      ifelse(df1$fertilizer == "B", "50", "0")
    )

    w <- pcv.emd(df1,
      cols = "hue_frequencies", reorder = c("fertilizer", "genotype"),
      mat = FALSE, plot = TRUE, parallel = 1
    )
  },
  error = function(err) {
    message(err)
  }
)

# Note on computational complexity
# This scales as O^2, see the plot below for some idea
# of the time for different input data sizes.
emdTime <- function(x, n = 1) {
  x^2 / n * 0.0023
}
plot(
  x = c(18, 36, 54, 72, 108, 135), y = c(0.74, 2.89, 6.86, 10.99, 26.25, 42.44),
  xlab = "N Input Images", ylab = "time (seconds)"
) # benchmarked test data
lines(x = 1:150, y = emdTime(1:150)) # exponential function

plot(
  x = 1:1000, y = emdTime(1:1000), type = "l",
  xlab = "N Input Images", ylab = "time (seconds)"
)
```

`pcv.joyplot` *Make Joyplots for multi value trait plantCV data*

Description

Make Joyplots for multi value trait plantCV data

Usage

```
pcv.joyplot(
  df = NULL,
  index = NULL,
  group = NULL,
  y = NULL,
  id = NULL,
  bin = "label",
  freq = "value",
  trait = "trait",
  fillx = TRUE
)
```

Arguments

`df`	Data frame to use. Long or wide format is accepted.
`index`	If the data is long then this is a multi value trait as a character string that must be present in 'trait'. If the data is wide then this is a string used to find column names to use from the wide data. In the wide case this should include the entire trait name (ie, "hue_frequencies" instead of "hue_freq").
`group`	A length 1 or 2 character vector. This is used for faceting the joyplot and identifying groups for testing. If this is length 1 then no faceting is done.
`y`	Optionally a variable to use on the y axis. This is useful when you have three variables to display. This argument will change faceting behavior to add an additional layer of faceting (single length group will be faceted, length 2 group will be faceted group1 ~ group2).
`id`	Optionally a variable to show the outline of different replicates. Note that ggridges::geom_density_ridges_gradient does not support transparency, so if fillx is TRUE then only the outer line will show individual IDs.
`bin`	Column containing histogram (multi value trait) bins. Defaults to "label".
`freq`	Column containing histogram counts. Defaults to "value"
`trait`	Column containing phenotype names. Defaults to "trait".
`fillx`	Logical, whether or not to use `ggridges::geom_density_ridges_gradient`. Default is T, if F then `ggridges::geom_density_ridges` is used instead, with arbitrary fill. Note that `ggridges::geom_density_ridges_gradient` may issue a message about deprecated ggplot2 features.

Value

Returns a ggplot.

Examples

```
library(extraDistr)
dists <- list(
  rmixnorm = list(mean = c(70, 150), sd = c(15, 5), alpha = c(0.3, 0.7)),
  rnorm = list(mean = 90, sd = 20),
  rlnorm = list(meanlog = log(40), sdlog = 0.5)
)
x_wide <- mvSim(
  dists = dists, n_samples = 5, counts = 1000,
  min_bin = 1, max_bin = 180, wide = TRUE
)
pcv.joyplot(x_wide, index = "sim", group = "group")
x_long <- mvSim(
  dists = dists, n_samples = 5, counts = 1000,
  min_bin = 1, max_bin = 180, wide = FALSE
)
x_long$trait <- "x"
p <- pcv.joyplot(x_long, bin = "variable", group = "group")
# we might want to display hues as their hue
p + ggplot2::scale_fill_gradientn(colors = scales::hue_pal(l = 65)(360))
x_long$group2 <- "example"
pcv.joyplot(x_long, bin = "variable", y = "group", fillx = FALSE)
```

---

pcv.net *Network analysis of a distance matrix*

Description

Easy igraph use with pcv.emd output

Usage

```
pcv.net(
  emd = NULL,
  meta = NULL,
  dissim = TRUE,
  distCol = "emd",
  filter = 0.5,
  direction = "greater"
)
```

Arguments

emd	A long dataframe as returned by pcv.emd. Currently this function is only made to work with dataframe output, not distance matrix output.
meta	Metadata to be carried from pcv.emd output into the network, defaults to NULL which will use all metadata. Type conversion will be attempted for these columns.
dissim	Logical, should the distCol be inverted to make a dissimilarity value?
distCol	The name of the column containing distances/dissimilarities. Defaults to "emd" for compatability with pcv.emd
filter	This can be either a numeric (0.5) in which case it is taken as a filter where only edges with values greater than or equal to that number are kept or a character string ("0.5") in which case the strongest X percentage of edges are kept. This defaults to 0.5 which does some filtering, although that should not be considered the best behavior for every setting. If this is NULL then your network will be almost always be a single blob, if set too high there will be very few nodes. Note that this filtering happens after converting to dissimilarity if dissim=TRUE.
direction	Direction of filtering, can be either "greater" or "lesser".

Value

Returns a list containing three elements: nodes: A dataframe of node data. edges: A dataframe of edges between nodes. graph: The network as an igraph object

Examples

```
library(extraDistr)
dists <- list(
  rmixnorm = list(mean = c(70, 150), sd = c(15, 5), alpha = c(0.3, 0.7)),
  rnorm = list(mean = 90, sd = 3)
)
x <- mvSim(
  dists = dists, n_samples = 5, counts = 1000,
  min_bin = 1, max_bin = 180, wide = TRUE
)
emd_df <- pcv.emd(x,
  cols = "sim", reorder = c("group"), mat = FALSE,
  plot = FALSE, parallel = 1
)
net <- pcv.net(emd_df, meta = "group")
net2 <- pcv.net(emd_df, meta = "group", filter = "0.9", direction = "lesser")
```

---

pcv.plsr *Run Partial Least Squares Regression on spectral data*

Description

Partial Least Squares Regression (plsr) is often used to analyze spectral data.

Usage

```
pcv.plsr(df, resps = NULL, spectra = NULL, train = 0.8, cv = 10, ...)
```

Arguments

df	Data frame containing metadata and spectral histogram data
resps	Vector of response variables.
spectra	Either one column name (in the case of long data) or a set of columns in the case of wide data. If a single character string is provided and it is not one of the column names then it is taken to be a pattern that will match some set of column names in the data to use (see examples).
train	Proportion of data to use as training data.
cv	Number of cross validation iterations.
...	Further arguments passed to caret::train.

Details

Note that columns that sum to 0 in the training or test data will be removed. This function also uses the 'pls' method from the pls package.

Value

a list of lists each with model performance, prediction target, model, plot, N components, and variable influence on projection components for each response variable.

Examples

```
if (rlang::is_installed("pls")) {
  dists <- list(
    rlnorm = list(meanlog = log(40), sdlog = 0.5),
    rlnorm = list(meanlog = log(60), sdlog = 0.35)
  )
  mv <- mvSim(
    dists = dists, n_samples = 100, counts = 1000,
    min_bin = 1, max_bin = 180, wide = TRUE
  )
  sv <- growthSim("logistic",
    n = 5, t = 20,
    params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
  )
  d <- cbind(sv, mv[, -1])
  # note that this requires the "pls" package to be installed.
  x <- pcv.plsr(df = d, resps = "y", spectra = grepl("^sim_", colnames(d)))
}
```

pcvrss-class *Class* pcvrss *for models specified in* pcvr.

Description

Models specified by growthSS or mvSS are represented by a pcvrss object, which contains the model type, formulas, starting values or priors, the data for the model to use, and the model backend to use.

Details

See methods(class = "pcvrss") for an overview of available methods.

Slots

formula The formula that will be used to fit the model.

prior Priors if the model is a Bayesian model (ie using the brms backend).

initfun Initialization function if the model is a Bayesian model.

df The data that will be used to fit the model.

family The model family, currently only used in the brms backend.

pcvrForm The formula that was specified in growthSS and used in other pcvr functions.

type The model backend.

model The name of the main growth formula.

call The call to growthSS or mvSS.

start Starting values for frequentist models.

taus Quantiles for nlrq/rq models.

See Also

growthSS, mvSS

pcv_theme *Default theme for ggplots made by pcvr functions.*

Description

Default theme for ggplots made by pcvr functions.

Usage

```
pcv_theme()
```

Value

A ggplot theme

Examples

```
ggplot2::ggplot() +
  pcv_theme()
```

`plotPrior` *Check priors used in ease of use brms functions*

Description

Check priors used in ease of use brms functions

Usage

```
plotPrior(priors, type = "density", n = 200, t = 25)
```

Arguments

`priors`	A named list of means for prior distributions. This takes the same input as the prior argument of `growthSS`. Alternatively, if given the output of growthSS this will preform a prior predictive check and return a plot from `growthPlot` of that check ignoring all other arguments. Note that all priors must be proper in that case (non-flat) and the fit is likely to be strange looking due to how thick tailed the default priors from `growthSS` are.
`type`	Either "density", the default, or a model as would be specified in `growthSS` or `growthSim` such as "logistic", "gompertz", "monomolecular", "exponential", "linear", "power law", "double logistic", or "double gompertz". If this is a model type then n draws from the prior will be simulated as growth trendlines and densities will be plotted on margins for some distributions.
`n`	Numeric, if type is a model then how many draws from the prior should be simulated?
`t`	Numeric, time passed to growthSim. Defaults to 25 (the growthSim default).

Value

A named list of plots showing prior distributions that `growthSS` would use, optionally with a plot of simulated growth curves using draws from those priors.

See Also

barg for Bayesian model reporting metrics, growthSim for simulating data using similar specification.

Examples

```
set.seed(123)
priors <- list("A" = c(100, 130), "B" = c(10, 8), "C" = c(0.2, 0.1))
plotPrior(priors)

plotPrior(priors, "gompertz")[[1]]
```

---

plotVIP *Plot Variable Influence on Projection*

Description

This function is used to visualize variable influence on projection (vip) from a plsr model.

Usage

```
plotVIP(plsrObject, i = 1, mean = FALSE, removePattern = ".*_")
```

Arguments

plsrObject	Output from pcv.plsr
i	An index from the plsrObject to use if the plsrObject contains models for several outcomes. Can be a name or a position. Defaults to 1.
mean	Logical, should the mean be plotted (TRUE) or should the components be shown individually (FALSE, the default).
removePattern	A pattern to remove to make the wavelength column into a numeric.

Value

A ggplot showing variable influence on projection

Examples

```
if (rlang::is_installed("pls")) {
  dists <- list(
    rlnorm = list(meanlog = log(40), sdlog = 0.5),
    rlnorm = list(meanlog = log(60), sdlog = 0.35)
  )
  mv <- mvSim(
    dists = dists, n_samples = 100, counts = 1000,
    min_bin = 1, max_bin = 180, wide = TRUE
  )
  sv <- growthSim("logistic",
    n = 5, t = 20,
    params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
  )
  d <- cbind(sv, mv[, -1])
```

```
  x <- pcv.plsr(df = d, resps = "y", spectra = grepl("^sim_", colnames(d)))
  plotVIP(x)
}
```

`print.pcvrss` *Print a* `pcvrss` *object.*

Description

Print a `pcvrss` object.

Usage

```
## S3 method for class 'pcvrss'
print(x, ...)
```

Arguments

`x`	An object of class `pcvrss` to method `summary` of `pcvrss`.
`...`	further arguments, passed to print.default.

See Also

`summary.pcvrss`

`print.pcvrsssummary` *Print a* `pcvrsssummary` *object.*

Description

Print a `pcvrsssummary` object.

Usage

```
## S3 method for class 'pcvrsssummary'
print(x, ...)
```

Arguments

`x`	An object of class `pcvrsssummary`.
`...`	further arguments, which are currently ignored.

See Also

`print.pcvrsssummary`

pwue *Calculate pseudo water use efficiency from phenotype and watering data*

Description

Rate based water use efficiency (WUE) is the change in biomass per unit of water metabolized. Using image based phenotypes and watering data we can calculate pseudo-WUE (pwue) over time. Here area_pixels is used as a proxy for biomass and transpiration is approximated using watering data. The equation is then $\frac{P_t - P_{t-1}}{W_{t_{end-1}} - W_{t_{start}}}$, where P is the phenotype and W is the weight before watering.

Absolute value based WUE is the amount of water used to sustain a plants biomass over a given period. The equation is then $\frac{P_t}{W_{t_{end-1}} - W_{t_{start}}}$

Usage

```
pwue(
  df,
  w,
  pheno = "area_pixels",
  time = "timestamp",
  id = "barcode",
  offset = 0,
  waterCol = "water_amount",
  method = "rate"
)
```

Arguments

df	Dataframe containing wide single-value phenotype data. This should already be aggregated to one row per plant per day (angles/rotations combined).
w	Watering data as returned from bw.water.
pheno	Phenotype column name, defaults to "area_pixels"
time	Variable(s) that identify a plant on a given day. Defaults to `c("barcode", "DAS")`.
id	Variable(s) that identify a plant over time. Defaults to `"barcode"`.
offset	Optionally you can specify how long before imaging a watering should not be taken into account. This defaults to 0, meaning that if a plant were watered directly before being imaged then that water would be counted towards WUE between the current image and the prior one. This argument is taken to be in seconds.
waterCol	Column containing watering amounts in `w`. This defaults to "watering_amount".

`method` Which method to use, options are "rate" and "abs". The "rate" method considers WUE as the change in a phenotype divided by the amount of water added. The "abs" method considers WUE as the amount of water used by a plant given its absolute size. The former is for questions more related to efficiency in using water to grow while the latter is more suited to questions about how efficient a plant is at maintaining size given some amount of water.

Value

A data frame containing the bellwether watering data joined to phenotype data with new columns for change in the phenotype, change in the pre-watering weight, and pseudo-water use efficiency (pWUE).

Examples

```
sim_water <- data.frame(
  "barcode" = "exampleBarcode1",
  "timestamp" = as.POSIXct(c(
    "2023-04-13 23:28:17 UTC",
    "2023-04-22 05:30:42 UTC",
    "2023-05-04 18:55:38 UTC"
  )),
  "DAS" = c(0.000000, 8.251675, 20.810660),
  "water_amount" = c(98, 12, -1)
)
sim_df <- data.frame(
  "barcode" = "exampleBarcode1",
  "timestamp" = as.POSIXct(c(
    "2023-04-13 23:28:17 UTC",
    "2023-04-22 05:30:42 UTC",
    "2023-05-04 18:55:38 UTC"
  )),
  "DAS" = c(0.000000, 8, 20),
  "area_pixels" = c(20, 1000, 1500)
)
pwue(
  df = sim_df, w = sim_water, pheno = "area_pixels",
  time = "timestamp", id = "barcode", offset = 0,
  waterCol = "water_amount", method = "rate"
)

pwue(
  df = sim_df, w = sim_water, pheno = "area_pixels",
  time = c("timestamp", "timestamp"), id = "barcode", offset = 0,
  waterCol = "water_amount", method = "abs"
)
```

`read.pcv` *Read in plantCV csv output in wide or long format*

Description

Read in plantCV csv output in wide or long format

Usage

```
read.pcv(
  filepath,
  mode = NULL,
  traitCol = "trait",
  labelCol = "label",
  valueCol = "value",
  reader = NULL,
  filters = NULL,
  awk = NULL,
  ...
)
```

Arguments

`filepath`	Path to csv file of plantCV output.
`mode`	NULL (the default) or one of "wide" or "long", partial string matching is supported. This controls whether data is **returned** in long or wide format. If left NULL then the output format will be the same as the input format.
`traitCol`	Column with phenotype names, defaults to "trait". This should generally not need to be changed from the default. This, labelCol, and valueCol are used to determine if data are in long format in their raw state (the csv file itself).
`labelCol`	Column with phenotype labels (units), defaults to "label". This should generally not need to be changed from the default. This is used with traitCol when `mode="wide"` to identify unique traits since some may be ambiguous (ellipseCenter.x vs ellipseCenter.y, bins of histograms, etc)
`valueCol`	Column with phenotype values, defaults to "value". This should generally not need to be changed from the default.
`reader`	The function to use to read in data, defaults to NULL in which case `data.table::fread` is used if filters are in place and `read.csv` is used otherwise. Note that if you use `read.csv` with filters in place then you will need to specify `header=FALSE` so that the piped output from awk is read correctly. If fread is too slow for your needs then `vroom::vroom()` may be useful.
`filters`	If a very large pcv output file is read then it may be desireable to subset it before reading it into R, either for ease of use or because of RAM limitations. The filter argument works with "COLUMN in VALUES" syntax. This can either be a character vector or a list of character vectors. In these vectors there needs to

be a column name, one of " in ", " is ", or " = " to match the string exactly, or "contains" to match with awk style regex, then a set of comma delimited values to filter that column for (see examples). Note that this and awk both use awk through `pipe()`. This functionality will not work on a windows system.

`awk` As an alternative to filters a direct call to awk can be supplied here, in which case that call will be used through `pipe()`.

`...` Other arguments passed to the reader function. In the case of 'fread' there are several defaults provided already which can be overwritten with these extra arguments.

Details

In plantCV version 4 the single value traits are returned in wide format from `json2csv` and the multi value traits are returned in long format. Briefly plantCV data was returned as one long table which sparked the emphasis in this function on reading data quickly and parsing it outside of R. With the current plantCV output these options are largely unnecessary. When data is read in using read.pcv the traitCol, valueCol, and labelCol arguments are checked to determine if the data is in long format. This is done to keep compatibility with interim versions of plantcv output where all outputs were in a single long format file.

With the current implementation and plantcv output you can read wide or long format files into wide or long format in R. Keep in mind that the 'mode' argument controls the format that will be returned in R, not the format that the data saved as in your csv file.

Value

Returns a data.frame in wide or long format.

Examples

```
tryCatch(
  {
    mv <- paste0(
      "https://media.githubusercontent.com/media/joshqsumner/",
      "pcvrTestData/main/pcv4-multi-value-traits.csv"
    )
    sv <- paste0(
      "https://raw.githubusercontent.com/joshqsumner/",
      "pcvrTestData/main/pcv4-single-value-traits.csv"
    )

    w2w <- read.pcv(sv, mode = "wide", reader = "fread")
    dim(w2w)

    w2l <- read.pcv(sv, mode = "long", reader = "fread")
    dim(w2l)

    l2w <- read.pcv(mv, mode = "wide", reader = "fread")
    dim(l2w)

    l2l <- read.pcv(mv, mode = "long", reader = "fread")
```

```
    dim(l2l)
  },
  error = function(e) {
    message(e)
  }
)
```

`read.pcv.3` *Read in plantCV csv from bellwether phenotyper style experiments analyzed with plantCV versions <4.*

Description

Read in plantCV csv from bellwether phenotyper style experiments analyzed with plantCV versions <4.

Usage

```
read.pcv.3(
  file = NULL,
  snapshotFile = NULL,
  designFile = NULL,
  metaCol = "meta",
  metaForm = "vis_view_angle_zoom_horizontal_gain_exposure_v_new_n_rep",
  joinSnapshot = "id",
  conversions = NULL,
  mode = "long",
  ...
)
```

Arguments

`file`	Path to the version 3 plantCV output containing phenotypes.
`snapshotFile`	path to the snapshot info metadata file, typically called SnapshotInfo.csv. This needs to have a column name corresponding to ‘joinSnapshot‘ (defaults to "id") which can be used to join the snapshot data to the phenotype data. Generally this joining will happen through a parsed section of the file path to each image present in the phenotype data. This means that including a duplicate name in ‘metaForm‘ will be overwritten by parsing image paths, so ‘metaForm‘ and ‘joinSnapshot‘ should not have duplicated names. If there is a timestamp column in the snapshot data then it will be converted to datetime (assuming a "Y-m-d H:M:S" format) and used to calculate days after starting (DAS) and hours.
`designFile`	path to a csv file which contains experimental design information (treatments, genotypes, etc) and which will be joined to phenotype and snapshot data through all shared columns.

metaCol	a column name from the phenotype data read in with the ‘file‘ argument. Generally for bellwether experiments this will correspond to an image path. The name is split on "/" characters with the last segment being taken and parsed into some number of sections based on ‘metaForm‘.
metaForm	A character string or character vector of column names to parse ‘metaCol‘ into. The number of names needs to match with length of ‘metaCol‘ when parsed. If a character string is provided then it is assumed to be underscore delimited, so do if you need underscores in a column name then use ‘c("column_one", "column_two",...)‘ instead of ‘column_one_column_two_...‘.
joinSnapshot	Column name create in phenotype data to use in joining snapshot data. By default this will attempt to make an "id" column, which is parsed from a snapshot folder in ‘metaCol‘ ("/shares/sinc/data/Phenotyper/SINC1/ImagesNew/**snapshot1403**/"). An error will be raised if this column is not present in the snapshot data.
conversions	A named list of phenotypes that should be rescaled by the value in the list. For instance, at zoom 1 ‘list(area = 13.2 * 3.7/46856)‘ will convert from pixels to square cm in the 5MP bellwether camera.
mode	The mode to read data in with through read.pcv. The default is "long" because this function is built for pcv3 output, which was generally a wider format to start with than pcv4 output.
...	Other arguments passed to `read.pcv`.

Value

Returns a dataframe potentially with several files merged into it.

Examples

```
tryCatch(
  {
    base_url <- "https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/"
    bw <- read.pcv.3(
      file = paste0(base_url, "pcv3Phenos.csv"),
      metaCol = NULL,
      reader = "fread"
    )
    bw <- read.pcv.3(
      file = paste0(base_url, "pcv3Phenos.csv"),
    metaCol = "meta", metaForm = "vis_view_angle_zoom_horizontal_gain_exposure_v_new_n_rep",
      joinSnapshot = "id",
      reader = "fread"
    )
    bw <- read.pcv.3(
      file = paste0(base_url, "pcv3Phenos.csv"),
      snapshotFile = paste0(base_url, "pcv3Snapshot.csv"),
      designFile = paste0(base_url, "pcv3Design.csv"),
    metaCol = "meta", metaForm = "vis_view_angle_zoom_horizontal_gain_exposure_v_new_n_rep",
      joinSnapshot = "id", conversions = list(area = 13.2 * 3.7 / 46856),
      reader = "fread"
    )
```

```
  },
  error = function(e) {
    message(e)
  }
)
```

`relativeTolerance` *Calculate relative tolerance of some phenotype(s) relative to control*

Description

Often in bellwether experiments we are curious about the effect of some treatment vs control. For certain routes in analysing the data this requires considering phenotypes as relative differences compared to a control. Note that the `conjugate` function can also be useful in considering the relative tolerance to stress between groups and that growth models are another suggested way to test relative tolerance questions.

Usage

```
relativeTolerance(
  df,
  phenotypes = NULL,
  grouping = NULL,
  control = NULL,
  controlGroup = NULL,
  traitCol = "trait",
  valueCol = "value"
)
```

Arguments

`df`	Dataframe to use, this can be in long or wide format.
`phenotypes`	A character vector of column names for the phenotypes that should be compared against control.
`grouping`	A character vector of column names that identify groups in the data. These groups will be calibrated separately, with the exception of the group that identifies a control within the greater hierarchy. Note that for levels of grouping where the control group does not exist the output will be NA.
`control`	A column name for the variable to be used to select the control observations. If left NULL (the default) then this will be taken as the first string in the group argument.
`controlGroup`	The level of the control variable to compare groups against.
`traitCol`	Column with phenotype names, defaults to "trait". This should generally not need to be changed from the default. If this and valueCol are present in colnames(df) then the data is assumed to be in long format.

valueCol — Column with phenotype values, defaults to "value". This should generally not need to be changed from the default.

Value

A dataframe with relative tolerance columns added.

Examples

```
f <- "https://raw.githubusercontent.com/joshqsumner/pcvrTestData/main/pcv4-single-value-traits.csv"
tryCatch(
  {
    sv <- read.pcv(
      f,
      reader = "fread"
    )
    sv$genotype <- substr(sv$barcode, 3, 5)
    sv$genotype <- ifelse(sv$genotype == "002", "B73",
      ifelse(sv$genotype == "003", "W605S",
        ifelse(sv$genotype == "004", "MM", "Mo17")
      )
    )
    sv$fertilizer <- substr(sv$barcode, 8, 8)
    sv$fertilizer <- ifelse(sv$fertilizer == "A", "100",
      ifelse(sv$fertilizer == "B", "50", "0")
    )

    sv <- bw.time(sv,
      plantingDelay = 0, phenotype = "area_pixels",
      cutoff = 10, timeCol = "timestamp", group = c("barcode", "rotation"), plot = FALSE
    )
    phenotypes <- colnames(sv)[19:35]
    phenoForm <- paste0("cbind(", paste0(phenotypes, collapse = ", "), ")")
    groupForm <- "DAS+DAP+barcode+genotype+fertilizer"
    form <- as.formula(paste0(phenoForm, "~", groupForm))
    sv <- aggregate(form, data = sv, mean, na.rm = TRUE)
    sv <- bw.outliers(sv,
      phenotype = "area_pixels",
      group = c("DAS", "genotype", "fertilizer"),
      plotgroup = c("barcode")
    )$data

    pixels_per_cmsq <- 42.5^2 # pixel per cm^2
    sv$area_cm2 <- sv$area_pixels / pixels_per_cmsq
    sv$height_cm <- sv$height_pixels / 42.5

    df <- sv
    phenotypes <- c("area_cm2", "height_cm")
    grouping <- c("fertilizer", "genotype", "DAS")
    controlGroup <- "100"
    control <- "fertilizer"

    rt <- relativeTolerance(df, phenotypes, grouping, control, controlGroup)
```

```
    head(rt)
    sapply(rt, function(c) sum(is.na(c)))
  },
  error = function(e) {
    message(e)
  }
)
```

`rqPlot` *Function to visualize* `quantreg::rq` *general additive growth models.*

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by `growthPlot`.

Usage

```
rqPlot(
  fit,
  form,
  df = NULL,
  groups = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

`fit`	A model fit, or list of model fits, returned by `fitGrowth` with type="nlrq" and model="gam".
`form`	A formula similar to that in `growthSS` inputs (or the `pcvrForm` part of the output) specifying the outcome, predictor, and grouping structure of the data as `outcome ~ predictor|individual/group`. If the individual and group are specified then the observed growth lines are plotted.
`df`	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for rq models.
`groups`	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
`timeRange`	An optional range of times to use. This can be used to view predictions for future data if the avaiable data has not reached some point (such as asymptotic size).

facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an rq general additive model's quantiles and optionally the individual growth lines.

Examples

```
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "gam", form = y ~ time | id / group,
  tau = c(0.25, 0.5, 0.75), df = simdf, start = NULL, type = "nlrq"
)
fits <- fitGrowth(ss)
rqPlot(fits, form = ss$pcvrForm, df = ss$df, groupFill = TRUE)
rqPlot(fits, form = ss$pcvrForm, df = ss$df, groups = "a", timeRange = 1:10)

ss <- growthSS(
  model = "gam", form = y ~ time | group,
  tau = c(0.5), df = simdf, start = NULL, type = "nlrq"
)
fit <- fitGrowth(ss)
rqPlot(fit, form = ss$pcvrForm, df = ss$df, groupFill = TRUE)
```

---

summary.pcvrss *Summarize a* pcvrss *object.*

Description

Summarize a pcvrss object.

Usage

```
## S3 method for class 'pcvrss'
summary(object, ...)
```

Arguments

object	An object of class pcvrss to method summary of pcvrss.
...	further arguments, passed to print.default.

survregPlot *Function to visualize* survival::survreg *models fit by* fitGrowth.

Description

Models fit using growthSS inputs by fitGrowth (and similar models made through other means) can be visualized easily using this function. This will generally be called by growthPlot.

Usage

```
survregPlot(
  fit,
  form,
  groups = NULL,
  df = NULL,
  timeRange = NULL,
  facetGroups = TRUE,
  groupFill = FALSE,
  virMaps = c("plasma")
)
```

Arguments

fit	A model fit returned by fitGrowth with type="nls".
form	A formula similar to that in growthSS inputs (or the pcvrForm part of the output) specifying the outcome, predictor, and grouping structure of the data as outcome ~ predictor|individual/group. If the individual and group are specified then the observed growth lines are plotted.
groups	An optional set of groups to keep in the plot. Defaults to NULL in which case all groups in the model are plotted.
df	A dataframe to use in plotting observed growth curves on top of the model. This must be supplied for nls models.
timeRange	Ignored, included for compatibility with other plotting functions.
facetGroups	logical, should groups be separated in facets? Defaults to TRUE.
groupFill	logical, should groups have different colors? Defaults to FALSE. If TRUE then viridis colormaps are used in the order of virMaps
virMaps	order of viridis maps to use. Will be recycled to necessary length. Defaults to "plasma", but will generally be informed by growthPlot's default.

Value

Returns a ggplot showing an survival model's survival function.

Examples

```
df <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- growthSS(
  model = "survival weibull", form = y > 100 ~ time | id / group,
  df = df, type = "survreg"
)
fit <- fitGrowth(ss)
survregPlot(fit, form = ss$pcvrForm, df = ss$df)
survregPlot(fit, form = ss$pcvrForm, df = ss$df, groups = "a")
survregPlot(fit,
  form = ss$pcvrForm, df = ss$df, facetGroups = FALSE,
  groupFill = TRUE, virMaps = c("plasma", "mako")
)
```

`testGrowth` *Hypothesis testing for fitGrowth models.*

Description

Hypothesis testing for fitGrowth models.

Usage

```
testGrowth(ss = NULL, fit, test = "A")
```

Arguments

`ss`	A list output from growthSS. This is not required for nls, nlme, and brms models if `test` is given in `brms::hypothesis` style as a written statement.
`fit`	A model (or list of nlrq models) output from fitGrowth. For brms models this can also be a data.frame of draws.
`test`	A description of the hypothesis to test. This can take two main forms, either the parameter names to vary before comparing a nested model ("A", "B", "C") using an anova or a hypothesis test/list of hypothesis tests written as character strings. The latter method is not implemented for `nlrq` models. If this is a vector of parameters to test in the model then they should be parameters which vary by group in your original model and that you want to test against a null model where they do not vary by group. Alternatively for nlrq models this can be a comparison of model terms written as `"group_X|tau|par - group_Y|tau|par"`, which uses a fat tailed T distribution to make comparisons on the means of each quantile estimate. For GAMs these tests compare the model with splines either by group or interacting with group to a model that ignores the grouping in the data. If this

is a list of hypothesis tests then they should describe tests similar to "A.group1 - A.group2*1.1" and can be thought of as contrasts. For brms models the "test" argument is passed to brms::hypothesis, which has extensive documentation and is very flexible. Note that for survreg the `survival::survdiff` function is used so fewer hypothesis testing options are available and flexsurv models are tested using contrasts via `flexsurv::standsurv`.

Details

For nls and nlme models an anova is run and returned as part of a list along with the null model. For nlrq models several assumptions are made and a likelihood ratio test for each tau is run and returned as a list.

Value

A list containing an anova object comparing non-linear growth models and the null model.

See Also

growthSS and fitGrowth for making compatible models, growthPlot for hypothesis testing on compatible models.

Examples

```
set.seed(123)
simdf <- growthSim("logistic",
  n = 20, t = 25,
  params = list("A" = c(200, 160), "B" = c(13, 11), "C" = c(3, 3.5))
)
ss <- suppressMessages(growthSS(
  model = "logistic", form = y ~ time | id / group,
  df = simdf, type = "nlrq"
))
fit <- fitGrowth(ss)
testGrowth(ss, fit, "A")
testGrowth(ss, fit, "a|0.5|A > b|0.5|A")

ss2 <- suppressMessages(growthSS(
  model = "logistic", form = y ~ time | id / group,
  df = simdf, type = "nls"
))
fit2 <- fitGrowth(ss2)
testGrowth(ss2, fit2, "A")$anova
coef(fit2) # check options for contrast testing
testGrowth(ss2, fit2, "A1 - A2*1.1")
```

Index