Package 'mosaicCalc'

March 16, 2024

Type Package

Title R-Language Based Calculus Operations for Teaching

Description Software to support the introductory *MOSAIC Calculus* textbook <https://www.mosaic-web.org/MOSAIC-Calculus/>), one of many data- and modeling-oriented educational resources developed by Project MOSAIC (<https://www.mosaic-web.org/>). Provides symbolic and numerical differentiation and integration, as well as support for applied linear algebra (for data science), and differential equations/dynamics. Includes grammar-of-graphics-based functions for drawing vector fields, trajectories, etc. The software is suitable for general use, but intended mainly for teaching calculus.

Version 0.6.1

Depends R (>= 3.5.0), mosaic, mosaicCore (>= 0.9.2),

Imports cubature, Deriv, dplyr, ggformula, ggplot2, glue, grDevices, MASS, Matrix, metR (>= 0.11.0), orthopolynom, plotly, rlang, Ryacas, sp, stats, tibble

Suggests testthat, knitr, palmerpenguins, rmarkdown, mosaicData

Author Daniel T. Kaplan <kaplan@macalester.edu>, Randall Pruim <rpruim@calvin.edu>, Nicholas J. Horton <nhorton@amherst.edu>

Maintainer Daniel Kaplan <kaplan@macalester.edu>

VignetteBuilder knitr

NeedsCompilation no

License GPL (>= 2)

LazyLoad yes

LazyData yes

URL https://github.com/ProjectMOSAIC/mosaicCalc

BugReports https://github.com/ProjectMOSAIC/mosaicCalc/issues

RoxygenNote 7.2.1

Encoding UTF-8

Repository CRAN

Date/Publication 2024-03-16 07:10:06 UTC

R topics documented:

`Aquaman` *Box office from the movie Aquaman*

Description

Earnings each weekend after the release of the movie *Aquaman*

Usage

```
Aquaman
```

Format

A data frame with 20 rows and variables `Date`, `Earnings` (in USD), and the count of the `Weekend`.

Source

Beth Schaubroeck at USAFA. Scraped from https://www.boxofficemojo.com/release/rl3108800001/weekend/?ref_=bo_rl_tab.

argM *Find local extreme points*

Description

Find local extreme points

Usage

```
argM(tilde, domain)
```

Arguments

tilde	specification of a function as in makeFun()
domain	a domain to search in.

Details

End-points of the domain may be included in the output.

Value

A data frame with values for x, the function output at those values of x, and the concavity.

Examples

```
argM(x^2 ~ x, domain(x=-1:1))
```

basis_sets *Basis sets for for function approximation*

Description

These functions generate the mathematical functions for three different basis sets: Fourier (sines), Legendre (orthogonal polynomials), and Splines (low-order smooth approximation)

Usage

```
legendre_set(df = 3, left = -1, right = 1)

legendre_M(x, df, left = -1, right = 1)

ns_set(df = 3, left = -1, right = 1)

fourier_M(x, n, fperiod = NULL)
```

```
ns_M(x, df = NULL, knots = NULL, intercept = FALSE, Boundary.knots = range(x))

fourier_set(df, left = -1, right = 1)
```

Arguments

df	number of basis functions to construct
left	number giving left-hand boundary of the interval
right	number giving right-hand boundary of the interval
x	inputs at which to evaluate the functions (in the _M functions)
n	number of fourier components to generate
fperiod	number giving the fundamental period length for the Fourier basis
knots	breakpoints that define the spline. The default is no knots; together with the natural boundary conditions this results in a basis for linear regression on x. Typical values are the mean or median for one knot, quantiles for more knots. See also Boundary.knots.
intercept	if TRUE, an intercept is included in the basis; default is FALSE.
Boundary.knots	boundary points at which to impose the natural boundary conditions and anchor the B-spline basis (default the range of the data). If both knots and Boundary.knots are supplied, the basis parameters do not depend on x. Data can extend beyond Boundary.knots

Details

For each basis, there are two different forms for the generating functions. Names ending in _set create a set of functions with arguments x and n, where integer n provides an index into the set. The same names with a _M suffix produce a model matrix corresponding to a specified set of x values. These are useful with lm() and similar model-building functions in the same way that poly() and ns() are useful. (ns_M() is just an alias for splines::ns().) Like poly() and ns(), the _M suffix functions do *NOT* include an intercept column.

Value

The _M functions return a model matrix. The _set functions return a function with arguments x and n. The integer n specifies which function to use, while x is the set of values at which to evaluate that function.

Blob1 *Shapes used in moment of inertia calculations*

Description

These are used in a handful of examples in the *MOSAIC Calculus* text.

Usage

```
Blob1

Blob2

Blob3

Blob4
```

Format

Data frames with coordinates `x` and `y` as columns

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 200 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 200 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 200 rows and 2 columns.

Body_fat *Measurements of body-fat percentage and related quantities*

Description

It's hard to measure body fat. This data frame records a direct measure of body-fat percentage along with easier-to-make measurements. The question of interest in collecting the data is whether there is a way to estimate body-fat percentage from the easier-to-measure quantities.

Usage

```
Body_fat
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 252 rows and 14 columns.

---

bounds *Specify a domain over which a function is to be graphed*

Description

domain() is used with slice_plot(), contour_plot(), or interactive_plot() to describe the plotting region. There is a standard syntax for domain() (see the first example) but there are also shortcuts. In the shorthand syntaxes, you can but don't have to specify the name of the input. If it's not specified, the plotting programs will try to do something sensible. But better to specify the names explicitly.

Usage

```
bounds(...)

domain(...)
```

Arguments

...	One or more expression of the form `x = -5:5`

Details

The colon operator is masked so that, for instance, `x = 0.5:1.3` literally means "0.5 to 1.3", and not just 0.5 as the base colon operator would give.

Value

a list with one component for each element in ...

Examples

```
## Not run:
contour_plot(x / y ~ x + y, bounds(x=-5:5, y=1:4))
slice_plot(x^2 ~ x, bounds(x = 2.5:4.2)) # overrides colon operator
slice_plot(x^2 ~ x, bounds(x = c(2.5, 4.2)))
slice_plot(x^2 ~ x, bounds(x = 1 %pm% 0.5))

## End(Not run)
```

---

box_set *Evenly spaced samples across a one- or two-dim domain*

Description

This function breaks up a domain in 1- or 2- dimensions into evenly spaced samples. It returns a data frame of the position of the samples, each of which can be considered to correspond to a Riemann bin for the purposes of integration.

Usage

```
box_set(tilde = NULL, domain, n = 10, sum = FALSE, dx = NULL)
```

Arguments

tilde	A tilde expression specifying the function to be evaluated on the domain.
domain	Either a one- or two-dimensional domain in the same format as for slice_plot() or contour_plot(), or a data frame with two columns specifying the coordinates of a polygon defining the area.
n	the number of divisions along each of the x- and y-directions. Can be a vector of length 2 giving different numbers for the x and for the y directions.
sum	If TRUE carry out the integral and return the numerical result.
dx	An alternative way of specifying the box size directly. Same for all dimensions.

Value

By default, a data frame listing the location of the samples, the .output. value of the given function at those samples, the spatial extent of the samples (that is, dx for one-dimension or dx and dy for two dimensions. There is also a dA giving the dx or dx*dy depending on the dimension). If sum=TRUE, then returns the sum of .output * dA, that is, the estimate of the integral of the function over the domain.

Examples

```
box_set(x*y ~ x & y, domain(x=0:1, y=0:1), n = 4)
# approximation to the variance of a uniform [0,1] distribution
box_set((x-.5)^2 ~ x, domain(x=0:1), n=100, sum=TRUE)
# a polygon
poly <- tibble(x = c(1:9, 8:1), y = c(1, 2*(5:3), 2, -1, 17, 9, 8, 2:9))
boxes <- box_set(1 ~ x & y, poly, dx = 1)
gf_polygon(y ~ x, data = poly, color="blue", fill="blue", alpha=0.2) %>%
  gf_rect((y - dy/3) + (y + dy/3) ~ (x - dx/3) + (x + dx/3),
  data = boxes)
# area inside polygon
box_set(1 ~ x & y, poly, n=100)
```

Boyle *Robert Boyle's pressure vs volume measurements*

Description

Pressure versus volume of air at a constant temperature as collected by Robert Boyle. Units are arbitrary.

Usage

```
Boyle
```

Format

A data frame with 7 rows and numerical variables pressure and volume.

Details

Robert Boyle (1627-1691) was a founder of modern chemistry and the scientific method in general. As any chemistry student already knows, Boyle sought to understand the properties of gasses. His results are summarized in *Boyle's Law*. The data frame `Boyle` contains two variables from one of Boyle's experiments as reported in his lab notebook: pressure in a bag of air and volume of the bag. The units of pressure are mmHg and the units of volume are cubic inches.

Source

Boyle's notebooks are preserved at the Royal Society in London. The data in the `Boyle` data frame have been copied from `https://chem.libretexts.org/Courses/University_of_California_Davis/UCD_Chem_002A/`

`Cello` *Short recordings of a cello and a violin*

Description

Only one note is involved.

- `Cello`: a complete note
- `Violin`: a complete note
- `Cello_seg`: a brief snipet of sound after the initial transient
- `Violin_seg`: as in `Cello_seg`

Usage

```
Cello

Cello_seg

Violin

Violin_seg
```

Format

Data frames (for convenience) with many rows and these variables:

- `y` the amplitude of the sound wave (arbitrary units)
- `t` the time of the sample in seconds.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 3616 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 43776 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 3638 rows and 2 columns.

Chirps *Cricket chirp rate and temperature*

Description

The rate at which crickets chirp is related to the ambient temperature. These data are motivated and described at https://www.globe.gov/explore-science/scientists-blog/archived-posts/sciblog/index.html_p=45.html

Usage

```
data(Chirps)
```

Format

A data frame with 55 rows

- `date` Which year the estimate is for.
- `time` in 24-hour format
- `chirps` the number of chirps per 15 seconds
- `temperature` in degrees F

contour_plot *Contour plots of functions of two variables*

Description

Creates a ggplot2-compatible contour plot of a function of two variables.

Usage

```
contour_plot(
  ...,
  npts = 100,
  labels = TRUE,
  filled = TRUE,
  contours_at = NULL,
  n_contours = 10,
  n_fill = 50,
  alpha = 1,
  fill_alpha = 0.1,
  label_alpha = 1,
  label_placement = 0.5,
  contour_color = "blue",
  label_color = contour_color,
```

```
  skip = 1,
  guide = FALSE,
  guide_title = "output",
  color_scale = scale_color_viridis_c(),
  fill_scale = scale_fill_viridis_d()
)
```

Arguments

`...`	Canonical first "three" arguments: `[Previous layer]`, tilde expression, domain
`npts`	Integer number of points on each axis at which to evaluate the function
`labels`	Logical flag: label the contours
`filled`	Logical flag: fill between the contours
`contours_at`	Vector of numbers. Contours will be drawn at these levels of the output.
`n_contours`	hint at number of contours to show
`n_fill`	Integer number of points for calculating fill
`alpha`	Transparency of contours in `[0-1]`
`fill_alpha`	Transparency of fill
`label_alpha`	Likewise, for contour labels
`label_placement`	A number, between 0 and 1, suggesting where to place the contour labels. Default: 0.5. This can be useful when there are two contour layers in one plaot.
`contour_color`	Set to a string, e.g. `"blue"` for contours to be drawn in a fixed color
`label_color`	Defaults to `contour_color`. Can be set to a
`skip`	Small integer. Skip this many contours between labeled contours
`guide`	Logical flag, whether to show a color guide.
`guide_title`	Character string for title of guide (if any)
`color_scale`	See ggplot2 color scales.
`fill_scale`	See ggplot2 fill scales

Value

ggplot2 graphics layers

Examples

```
## Not run:
contour_plot(sin(0.2*x*y) ~ x & y, domain(x=-10.3:4.5, y=2:7.5))
contour_plot(x + y ~ x & y)
contour_plot(sin(0.2*x*y)) # but better to use tilde expression

## End(Not run)
```

Covid_US — *COVID data from the first half of the pandemic*

Description

For documentation, see https://covid19datahub.io/articles/doc/data.html

Usage

```
Covid_US
```

Format

An object of class grouped_df (inherits from tbl_df, tbl, data.frame) with 575 rows and 36 columns.

Details

See Guidotti, E., Ardia, D., (2020), "COVID-19 Data Hub", Journal of Open Source Software 5(51):2376, doi: 10.21105/joss.02376.

CPUs — *Characteristics of computer central processing unit chips over the decades*

Description

Since single-chip central processing units were introduced in the early 1970s, performance has increased exponentially. Around 1965, Gorden E. Moore, a co-founder of the integrated circuit giant Intel, postulated that capacity of chips, measured as the number of transistors, doubles approximately every two years. Known as Moore's Law, this pattern has held true for decades.

Usage

```
data(CPUs)
```

Format

A data frame with 128 cases, each a computer CPU.

- processor Model name/number of the processor
- transistors Number of transistors
- year Year of market introduction of the CPU
- designer The company that introduced the CPU
- process The width of circuit elements, in nm.
- area Area of the chip in mm-squared.
- density The number of transistors per unit area, simply transistors/area.

Source

Dr. Joe Eichholz

`create_num_antiD` *Create a numerical anti-derivative function which can be called with one or many values of the w.r.t. input*

Description

This will typically be called directly from `antiD()` when an integral can't be handled symbolically.

Usage

```
create_num_antiD(tilde, ..., lower = NULL, .tol = 1e-04)
```

Arguments

`tilde`	Tilde expression for the function to be anti-differentiated. Right-hand side will be the w.r.t. variable
`...`	arguments and parameters to the function described by `tilde`
`lower`	Optional lower bound of integration. Useful to avoid domain problems with the function being integrated, but not generally needed.
`.tol`	Numerical tolerance for the integration

Value

a function with the w.r.t. variable as the first argument. The function is a wrapper around numerical integration routines.

`D` *Derivative and Anti-derivative operators*

Description

Operators for computing derivatives and anti-derivatives as functions.

Usage

```
D(tilde, ...)

antiD(tilde, ..., lower.bound = 0, force.numeric = FALSE, .tol = 1e-04)
```

Arguments

tilde	A tilde expression. The right side of a formula specifies the variable(s) with which to carry out the integration or differentiation. On the left side should be an expression or a function that returns a numerical vector of the same length as its argument. The expression can contain unbound variables. Functions will be differentiated as if the formula `f(x) ~ x` were specified but with `x` replaced by the first argument of `f`.
...	Default values to be given to unbound variables in the expression `expr`. See examples.#' Note that in creating anti-derivative functions, default values of "from" and "to" can be assigned. They are to be written with the name of the variable as a prefix, e.g. `y.from`.
lower.bound	for numerical integration only, the lower bound used
force.numeric	If `TRUE`, a numerical integral is performed even when a symbolic integral is available.
.tol	Tolerance for numerical integration. Most users do not need this.

Details

`D()` attempts to find a symbolic derivative for simple expressions, but will provide a function that is a numerical derivative if the attempt at symbolic differentiation is unsuccessful. The symbolic derivative can be of any order (although the expression may become unmanageably complex). The numerical derivative is limited to first or second-order partial derivatives (including mixed partials). `antiD()` will attempt simple symbolic integration but if it fails it will return a numerically-based anti-derivative.

`antiD()` returns a function with the same arguments as the expression passed to it. The returned function is the anti-derivative of the expression, e.g., `antiD(f(x)~x) -> F(x)`. To calculate the integral of `f(x)`, use `F(to) - F(from)`.

Value

For derivatives, the return value is a function of the variable(s) of differentiation, as well as any other symbols used in the expression. Thus, `D(A*x^2 + B*y ~ x + y)` will compute the mixed partial with respect to x then y (that is, $\frac{d^2 f}{dy\ dx}$). The returned value will be a function of x and y, as well as A and B. In evaluating the returned function, it's best to use the named form of arguments, to ensure the order is correct.

a function of the same arguments as the original expression with a constant of integration set to zero by default, named "C", "D", ... depending on the first such letter not otherwise in the argument list.

Examples

```
D(sin(t) ~ t)
D(A*sin(t) ~ t )
D(A*sin(2*pi*t/P) ~ t, A=2, P=10) # default values for parameters.
f <- D(A*x^3 ~ x + x, A=1) # 2nd order partial -- note, it's a function of x
f(x=2)
f(x=2,A=10) # override default value of parameter A
g <- D(f(x=t, A=1)^2 ~ t)  # note: it's a function of t
```

```
g(t=1)
gg <- D(f(x=t, A=B)^2 ~ t, B=10)  # note: it's a function of t and B
gg(t=1)
gg(t=1, B=100)
f <- makeFun(x^2~x)
D(f(cos(z))~z) #will look in user functions also
antiD( a*x^2 ~ x, a = 3)
G <- antiD( A/x~x ) # there will be an unbound parameter in G()
G(2, A=1) # Need to bound parameter. G(2) will produce an error.
F <- antiD( A*exp(-k*t^2 ) ~ t, A=1, k=0.1)
F(t=Inf)
one = makeFun(1 ~ x)
by.x = antiD(one(x) ~ x)
by.xy = antiD(by.x(sqrt(1-y^2)) ~ y)
4 * by.xy(y = 1) # area of quarter circle
```

`df2matrix` *Construct a model matrix from data as if by hand*

Description

The *MOSAIC Calculus* course includes a block on linear algebra. As part of this block, we want to cover creating model matrices and evaluating the model constructed from them. One way to do this is to introduce `lm()` and the domain specific language for specifying model terms. However, that introduces oddities. For instance, `lm(mpg ~ hp + hp^2, data = mtcars)` does NOT add the quadratic term to the model matrix. Also, `lm()` doesn't produce a residual or the model vector (ahem ... the fitted values) in the form of vectors. That's fine if you're teaching statistical modeling, but in the *MOSAIC Calculus* linear algebra block we are not teaching statistics, but the mathematical pre-requisites to understanding statistics.

Usage

```
df2matrix(..., data = NULL)
```

Arguments

`...`	Expressions, written in terms of the variable names in the data frame, that are to be collected into the model matrix.
`data`	The data frame from which to which the variable names will be bound.

Details

Specifically for *MOSAIC Calculus*, we have added this `df2matrix()` function. It serves much the same purpose as `cbind()`, that is, collecting vectors into a matrix. But is has two additional features:

1. It has a `data=` argument so that it can refer to a data frame.
2. It names the columns of the matrix with the code that was used to create each column.

The consequence of (2) is that the `x` vector produced by `qr.solve()` will have the same names as the matrix. That helps in interpreting `x`. But those names can also be used by `makeFun()` to generate a function from `x`.

`1` is a good form in which to write the intercept term.

Value

a matrix

Examples

```
A <- df2matrix(1, disp, log(hp), sin(cyl)*sqrt(hp), data = mtcars)
b <- df2matrix(mpg, data = mtcars)
x <- qr.solve(A, b)
f <- makeFun(x)
f(hp=3, disp=2, cyl=4)
```

`EbolaAll` *Case numbers in an Ebola outbreak in 2014*

Description

In December 2013, an 18-month-old boy from a village in Guinea suffered fatal diarrhea. Over the next months a broader outbreak was discovered, and in mid-March 2014, the Pasteur Institute in France confirmed the illness as Ebola-Virus Disease caused by the Zaire ebolavirus. Although the outbreak was first recognized in Guinea, it eventually encompassed Liberia and Sierra Leone as well. By July 2014, the outbreak spread to the capitals of all three countries.

Usage

```
EbolaAll

EbolaGuinea
```

Format

A dataframe with 182 rows. Each row is a daily report during a period of over 18 months during 2014 and 2015. Each report gives the number of new cases and disease-related deaths since the last report in each or three countries: Sierra Leon, Liberia, and Guinea. These values have been calculated from the raw, cumulative data. The data have been scrubbed to remove obvious errors.

- `Date`: Date when the World Health Organization issued the report
- `Gcases`: Number of new cases in Guinea
- `Gdeaths`: Number of new deaths in Guinea
- `Lcases`: Number of new cases in Liberia
- `Ldeaths`: Number of new deaths in Liberia

- SLcases: Number of new cases in Sierra Leone
- SLdeaths: Number of new deaths in Sierra Leone
- TotCases: Cumulative number of cases across all three countries
- TotDeaths: Cumulative number of deaths across all three countries Added in EbolaGuinea
- Days: When the report was issued in terms of a count of days from the initial report.
- G7Rcases: Number of new cases in Guinea averaged across 7 reports
- G7Rdeaths: Number of new deaths in Guinea averaged across 7 reports

An object of class spec_tbl_df (inherits from tbl_df, tbl, data.frame) with 182 rows and 6 columns.

Details

- EbolaAll gives daily reports for Guinea, Liberia, and Sierra Leone.
- EbolaGuinea contains just the Guinea data, but adds columns containing a 7-day moving average.

Source

US Centers for Disease Control (CDC).

Effective_oxygen *Effective amount of oxygen available at different altitudes*

Description

Air pressure changes with altitude and therefore the amount of oxygen available changes as well.

Usage

```
data(Effective_oxygen)
```

Format

A data frame with 30 rows

- altitude‘ In feet above sea level
- oxygen‘ Effective oxygen as a proportion of ai-

Source

https://wildsafe.org/resources/ask-the-experts/altitude-safety-101/oxygen-levels/

Engines *Characteristics of various internal combustion engines*

Description

Internal combustion engines have been built for a variety of purposes ranging from propelling small model airplanes to powering trucks to driving the propellers of giant ships. This data frame gives the characteristics of a wide size range of such engines.

Usage

```
data(Engines)
```

Format

A data frame with 39 cases, each of which is an internal combustion engine, with observations on the following variables.

- `mass` In pounds
- `ncylinder` Number of cylinders
- `stroke` The length of the stroke made by a piston
- `strokes` Whether the engine cycle is 2-stroke or 4-stroke.
- `displacement` In cc
- `bore` Diameter of the piston
- `BHP` The power generated by the engine. One BHP is the same as 745.7 Watts.
- `RPM` The speed, in revolutions per second, at which the engine generates the listed BHP.

Source

McMahon, Thomas A., and John Tyler Bonner. On Size and Life. New York: Scientific American Library, 1983. pp. 60-61

first_three_args *Handle the first three arguments of graphics functions*

Description

This function is intended for package developers, not end-users. Canonically, `mosaicCalc` functions that produce layerable graphics have three initial arguments in a specific order: (1) a previous gg layer, (2) a tilde expression, and (3) a domain. But either (1) or (3) can be missing. `first_three_args()` translates a leading ... argument into the list of the canonical three initial arguments, returning them as components of a list. In addition, there may be additional arguments in ... that specify other aspects of the plot, e.g. color.

Usage

```
first_three_args(..., two_tildes = FALSE)
```

Arguments

...	unnamed arguments to be translated into a list with the three canonical arguments and any other arguments not named explicitly in the parent function definition.
two_tildes	if TRUE then look for the first FOUR arguments, the middle two of which will be tilde expressions.

Details

In constructing a mosaicCalc graphics layer, the function (e.g. `slice_plot()` or `contour_plot()`) whould have ... as its first argument. Intercept that ... with `first_three_args()` to extract the first three canonical arguments as components `gg`, `tilde`, and `domain` of a list. Any remaining arguments in ... will be placed in the `dots` component.

---

fitSpline *Find zeros of a function*

Description

This is defined in the `mosaic package`: See `fitSpline`

Fly_ball *Trajectory of a fly ball in baseball*

Description

The trajectory of a fly ball as calculated by a sophisticated model involving drag, spin, and such. The physics and mathematics are described at http://baseball.physics.illinois.edu/TrajectoryAnalysis.pdf

Usage

```
Fly_ball
```

Format

A data frame with 552 rows.

- `t` - (seconds). Runs from 0 to 5.51 s
- `y` - horizontal position (feet)
- `z` - vertical position (feet)

Details

The ball has a mass of 5.125 ounces and a circumference of 9.125 inches The speed off the bat was 103 mph at a launch angle of 27.5 degrees from the horizontal. The ball was hit with backspin at 2500 rpm by a right-handed batter. Ambient temperature 70 deg F with a relative humidity of 50% and a barometric pressure of 29.92 inHg. The field was at an elevation of 15 feet.

The position of the ball was recorded every 0.01 seconds, for 5.5 seconds altogether. If you prefer to work with continuous functions of time, you can construct them. See the examples.

Source

http://baseball.physics.illinois.edu/TrajectoryCalculator-new-3D.xlsx

Examples

```
# Constructing continuous functions of time
ball_z <- mosaic::connector(z ~ t, data = Fly_ball)
ball_y <- mosaic::connector(y ~ t, data = Fly_ball)
```

---

gradient_plot *Plot a vector field*

Description

Plot a vector field

Usage

```
gradient_plot(..., npts = 20, color = "black", alpha = 0.5, transform = sqrt)

vectorfield_plot(
  ...,
  npts = 20,
  color = "black",
  alpha = 0.5,
  transform = sqrt,
  env = NULL
)
```

Arguments

...	(Optional: a previous graphics layer), tilde expressions for horiz and vertical component of vectors, a domain (if not inherited from the graphics layer). The tilde expressions should be in the same style as for `makeODE()`. See details.
npts	number of arrows to draw in one row or column of the field.
color	character string specifying the color of the arrows.

alpha	transparency of the arrots
transform	controls the relative length of the arrows. See details.
env	Not for end-users. Handles details of where things are defined.

Details

There will be one tilde expression for the horizontal component of the vector and another tilde expression for the vertical component. Suppose the horizontal axis is called u and the vertical axis is called v, as would be established by a bounds specification like `bounds(u=-1:1, v=-1:1)`. Then the horizontal tilde expression **must** have a left side called `u ~`. Similarly, the vertical tilde expression will have a left side called `v ~`. On the right side of the tilde expressions will go the formulas for the respective components of the vectors, e.g. `u ~ sin(u-v)` and `v ~ v*u^2`.

Typically, the length of the arrows is not meaningful in the units of the horizontal or vertical axis. For instance, in a gradient plot of f(x,y), the axis is in units of x, but the gradient component has units of f(x,y)/x. Similarly for the flow of a differential equation. Nonetheless, the relative lengths of the arrows, one to another, does have meaning. In drawing the vector field, the arrows are scaled so that the longest one barely avoids contact with it's neighbors. This natural scaling has the disadvantage that it can be hard to discern the lengths of the shortest arrows, which often are near zero (as with the gradient near the argmax or argmin, or near a fixed point of a differential equation flow field). By default, the relative lengths of the arrow are transformed by sqrt, to make the long arrows shorter and therefore enable the sort arrows to be drawn somewhat longer. If you want the natural scaling instead, use `transform=I`. Or you might want to make the arrows even more similar in length. Then use, for instance `transform=function(L) L^0.1`

Value

ggplot2 graphics layers

Examples

```
gradient_plot(x * sin(y) ~ x & y, bounds(x=-1:1, y=-1:1), transform=I)
vectorfield_plot(x ~ -y, y ~ x, bounds(x=-1:1, y=-1:1))
gf_label(0 ~ 0, label="center", color="red") %>%
vectorfield_plot(x ~ -y, y ~ x, bounds(x=-1:1, y=-1:1), transform=function(x) x^0.2 )
vectorfield_plot(u ~ sin(u-v), v ~ v*u^2, bounds(u=0:1, v=-1:1))
```

HDD_Minneapolis *Heating degree days in Minneapolis, Minnesota, USA*

Description

A "heating degree day" is a measure of weather coldness. It's defined to be the difference between the outdoor ambient temperature and 65 degrees F, but has a value of zero when the ambient temperature is above 65 degrees. This difference is averaged over time and multiplied by the number of days in the time period covered. The heating degree day is often used as a measure of the demand for domestic heating in a locale.

Usage

```
data(HDD_Minneapolis)
```

Format

A data frame `HDD_Minneapolis` with 1412 rows and 4 variables:

- year the year
- month the month
- hdd the number of heating degree days for that period.
- loc the location at which the temperature was measured. In the early years, this was downtown Minneapolis. Later, the site was moved to the Minneapolis/Saint-Paul International Airport.

Details

These data report monthly heating degree days. For teaching purposes, the data give an extreme example of how a relationship (hdd vs year) can be revealed by including a covariate (month). Although interest focusses on the change in temperature over the century the data cover, there is such regular seasonal variation that no systematic trend over the years is evident unless month is taken into account.

Examples

```
mod_1 <- lm(hdd ~ year, data = HDD_Minneapolis)
mod_2 <- lm(hdd ~ year + month, data = HDD_Minneapolis)
```

`Home_utilities` *Gas and electricity usage by a home in St. Paul, MN*

Description

Gas and electricity usage by a home in St. Paul, MN

Usage

```
data(Home_utilities)
```

Format

A data.frame object with one row for each month

- `month` - the month covered by the bill as a number
- `day` - the terminating day of the bill
- `year` - the year in which the month fell
- `temp` - average temperature over the entire month
- `kwh` - electricity usage in kilowatt-hours

- `ccf` - natural gas usage in cubic feet
- `thermsPerDay` - another measure of natural gas usage
- `dur` - duration of the period covered by the bill, in days (typically about 30)
- `totalbill` - dollar amount of the bill
- `gasbill`: - dollar amount of the natural gas part of the bill
- `elecbill` - dollar amount of the electricity bill
- `generatedKwh` - Energy generated by the photovoltaic system
- `notes` - miscellaneous comments on the month's bill or changes in the house

Details

A college mathematics professor teaching in St. Paul, Minnesota, USA collected these records of his utility bills starting in 1999. Starting in June 2022, solar photovoltaics were installed on the roof of the house. There are months missing, a few transcription errors, etc. **These data will be updated occasionally, as new bills come in.**

Source

Personal communication from the homeowner

---

inequality_constraint *Graphics for constraints*

Description

These functions are intended to annotate contour plots with constraint regions. Inequality constraints are shaded where the constraint is NOT satisfied. Equality constraints are shaded in a small region near where the constraint is satisfied.

Usage

```
inequality_constraint(
  previous = NULL,
  tilde,
  domain,
  npts = 100,
  fill = "blue",
  alpha = 1
)

equality_constraint(
  previous = NULL,
  tilde,
  domain,
  npts = 100,
  fill = "blue",
  alpha = 1
)
```

Arguments

previous	can be ignored by user. It supports the pipe syntax for layering graphics.
tilde	a tilde expression specifying the constraint. For an inequality constraint this should be a logical expression that is TRUE where the constraint is satisfied. For an equality constraint, the left-hand side of the tilde expression should be zero where the constraint IS satisfied.
domain	as in contour_plot(), the domain over which to graph the constraint
npts	a number specifying how finely to divide the domain in each direction. Default is 100, but this gives a discernably pixelated appearance to the shading. 200 or 300 is more appropriate for publication-quality graphics.
fill	the color to use for shading
alpha	the opacity of the shading

Value

ggplot2 graphics layers

Examples

```
inequality_constraint(x + y > 2 ~ y + x, domain(y=0:3, x=0:2))
equality_constraint(x + y - 2 ~ y + x, domain(y=0:3, x=0:2), npts=200, alpha=.3, fill="red")
```

---

infer_RHS *Utilities for formulas and graphics arguments*

Description

infer_RHS turns a one-sided formula into a two-sided formula suitable for makeFun(). formals_from_expression creates a "formals" list for creating a function. The list will have arguments in the canonical order.

Usage

```
infer_RHS(ex)

formals_from_expr(ex, others = character(0))
```

Arguments

ex	An expression as in quote(x^2) or the left side of a tilde expression.
others	Character string(s) with names of additional arguments to be included in the formals

inscribed_circle *Create a data frame for a circle marking the curvature of a function.*

Description

Create a data frame for a circle marking the curvature of a function.

Usage

```
inscribed_circle(ftilde, x0 = 0)
```

Arguments

ftilde	A tilde expression defining a function
x0	The value for the input at which the curvature is to be calculated.

Value

`inscribed_circle()` return a data frame with `x` and `y` components describing the inscribed circle located at the point $(x_0, f(x_0))$. Plot this with `gf_path(y ~ x)` `curvature_function()` returns a function that gives the curvature of the specified function for any input x. It's much like `D()` in the way it is used.

Integrate *Integrate a function*

Description

Calculates the definite integral of a function. That is, the result of the integration will be a number. There can be no free parameters in the function being integrated. (If you want free parameters, use `antiD()`.) `Integrate()` can handle integration over up to 3 variables.

Usage

```
Integrate(tilde, domain, ..., tol = 1e-05)
```

Arguments

tilde	A tilde expression describing the function.
domain	The domain over which to perform the integration
...	values assigned to free parameters in the tilde expression, e.g. a=1
tol	numerical tolerance for integration, see `stats::integrate()`.

Details

For functions constructed as a spline interpolant, `Integrate()` can handle more segments than `antiD()`. It's reasonable to do up to 2000 segments with `Integrate()`, whereas `antiD()` handles only about 100.

Examples

```
Integrate(dnorm(x) ~ x, domain(x=-2:2))
Integrate(dnorm(x, sd=sigma) ~ x, domain(x=-2:2), sigma=2)
Integrate(sqrt(1- x^2) ~ x, domain(x=-1:1)) # area of semi-circle
```

---

integrateODE *Integrate ordinary differential equations*

Description

A formula interface to integration of an ODE with respect to "t"

Usage

```
integrateODE(...)
```

Arguments

`...`	A dynamics object (see `makeODE()`) and/or arguments giving additional formulas for dynamics in other variables, assignments of parameters, assignments of initial conditions, the start and end times of the integration (through `domain()`), and the step size (through `dt=`).

Details

The equations must be in first-order form. Each dynamical equation uses a formula interface with the variable name given on the left-hand side of the formula, preceded by a `d`, so use `dx~-k*x` for exponential decay. All parameters (such as `k`) must be assigned numerical values in the argument list. All dynamical variables must be assigned initial conditions in the argument list. The returned value will be a list with one component named after each dynamical variable. The component will be a spline-generated function of `t`.

Value

a list with splined function of time for each dynamical variable

Examples

```
soln = integrateODE(dx~r*x*(1-x/k), k=10, r=.5, domain(t=0:20), x=1)
soln$x(10)
soln$x(30) # outside the time interval for integration
traj_plot(x(t)~t, soln, domain(t=0:10))
soln2 = integrateODE(dx~y, dy~-x, x=1, y=0, domain(t=0:10))
traj_plot(y(t)~t, soln2)
SIR <- makeODE(dS~ -a*S*I, dI ~ a*S*I - b*I, a=0.0026, b=.5, S=762, I=1)
epi = integrateODE(SIR, domain(t=0:20))
traj_plot(I(t) ~ t, epi)
```

---

is_in_domain *check whether a value is in a domain*

Description

A convenience function to see if a value is inside (or on the boundaries of) a domain or a set of domain segments. In the case of multiple segments, the check is whether `val` is within any of them.

Usage

```
is_in_domain(val, domain, ...)
```

Arguments

`val`	the values to be checked
`domain`	the domain
`...`	additional domains

Examples

```
is_in_domain(1:10, domain(x=5.5:8.5), domain(x=1:2))
is_in_domain(mtcars, domain(mpg=15:20, wt=2:4.5), domain(mpg=25:28))
```

Iterate *Iterate a function on an initial condition*

Description

Iterates a function a specified number of times on an initial condition.

Usage

```
Iterate(f = NULL, A = NULL, x0 = 0, n = 10, fargs = list())
```

Arguments

f	a function of one or more state variables which returns a vector containing those same state variables. Parameters to f(), if any, should be named and at the end of the argument list. State variables should **not** have default values.
A	As an alternative to f you can give a square matrix and iteration will be done on the system x[n+1] = A x[n]. In lieu of a square matrix you can also give a vector to be rendered into a square matrix rowwise.
x0	a vector with the numerical initial condition. There should be 1 component in x0 for each of the state variables in f().
n	an integer specifying the number of iterations
fargs	list containing values for numerical parameters to the function f()

Details

The function f can take one or more arguments. The first of these should represent the *state* of the dynamical system, e.g. x, or x and y, etc. At the end of the argument list to f can come named parameters. The length of the initial condition x0 must match the number of state arguments. Numerical values for parameters (if any) must be provided in the ... slot.

Value

A data frame with a column .i listing the iteration number and columns for each component of the initial condition. There will be n+1 rows, the first for the initial condition and the remaining for the n iterations.

Examples

```
Iterate(function(x, mu=3.5) mu*x*(1-x), x0=.232, n=10, list(mu=4)) # chaos
Iterate(function(x, y) c(x+y, x-y), x0 = c(1,1), n=10)
Iterate(function(x, y) c(x+y, x), x0=c(1,0), n=10) # fibonacci
Iterate(A = cbind(rbind(1, 1), rbind(1, 0)), x0=c(1,0), n=5) # fibonacci described by a matrix
```

Kepler *Kepler's calculation of the position of Mars*

Description

Astronomer Johannes Kepler (1571-1630) famously measured the position of the planet Mars. His interpretation of this data led substantially to Newton's theory of universal gravity. This data frame gives Kepler's measurements as a function of time, together with a modern reconstruction of the "actual" position of Mars at the time.

Usage

```
data(Kepler)
```

Format

A data frame with 28 rows and these variables:

- `time` - Interval in Earth days from 8:15am Greenwich time on 9 March 1584.
- `kepler.radius` - The radius of the orbit in AU (astronomical units. 1 AU is 93 million miles)
- `kepler.angle` - The "true anomoly" measured in radians
- `actual.radius` - Modern calculation of the above
- `actual.angle` - Modern calculation of the above

Details

The raw measurements (not included here) that Kepler used in his calculation were made by Tycho Brahe (1546-1601). Those raw measurements were of the angle of Mars with respect to Earth. Kepler estimated the orbital period of Mars to be 687 Earth days. (The current accepted value is 686.980 days.) Knowing the period, Kepler could find pairs of Earth days separated by multiples of the period. In each pair, the Earth would be in a different position, but Mars would be in the same position. Thus the distance of Mars from Earth could be estimated by triangulation.

The angle was not directly measured for each occasion. Instead, knowing the radius versus time Kepler was able to discern when Mars was at its greatest and closest distance to the Sun. The angle tells where Mars is along its orbit. An angle of 0 is the position when Mars is closest to the Sun. An angle of 3.14 is when Mars is farthest from the Sun.

Source

Drawn from from McLaughlin, Michael P. (1999) "A Tutorial on Mathematical Modelling" p. 21-23.

References

See `https://faculty.uca.edu/saustin/3110/mars.pdf` for a useful description of the estimation process.

`M2014F` *Mortality versus age for females in the US in 2014*

Description

These data are from the US Social Security Administration.

Usage

```
M2014F
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 120 rows and 5 columns.

Details

- `pmort` the probability of a person of the given age dying in 2014
- `nliving` traces a hypothetical population of 100,000 people through the mortality at each age.
- `died` for the hypothetical population the number of people at that age who survived to the age and then died at that age. Source: https://www.ssa.gov/oact/HistEst/PerLifeTables/2017/PerLifeTables2017.html

---

makeODE *Create a dynamics object for use in* `integrateODE()` *and the ODE graphics*

Description

An ODE object consists of some dynamics, initial conditions, parameter values, time domain, etc.

Usage

```
makeODE(...)
```

Arguments

`...`	The components of an ODE and/or a set of other ODE objects

Value

a list containing various functions and values specifying the ODE.

Examples

```
SIR <- makeODE(dS~ -a*S*I, dI ~ a*S*I - b*I, a=0.0026, b=.5, S=762, I=1)
soln <- integrateODE(SIR, domain(t=0:20))
```

---

numD *Numerical Derivatives*

Description

Constructs the numerical derivatives of mathematical expressions

Usage

```
numD(tilde, ..., .h = NULL)
```

Arguments

tilde	a mathematical expression (see examples and plotFun)
...	additional parameters, typically default values for mathematical parameters
.h	numerical step size to enforce.

Details

Uses a simple finite-difference scheme to evaluate the derivative. The function created will not contain a formula for the derivative. Instead, the original function is stored at the time the derivative is constructed and that original function is re-evaluated at the finitely-spaced points of an interval. If you redefine the original function, that won't affect any derivatives that were already defined from it. Numerical derivatives, particularly high-order ones, are unstable. The finite-difference parameter .h is set, by default, to give reasonable results for first- and second-order derivatives. It's tweaked a bit so that taking a second derivative by differentiating a first derivative will give reasonably accurate results. But, if taking a second derivative, much better to do it in one step to preserve numerical accuracy.

Value

a function implementing the derivative as a finite-difference approximation. This has a second argument, .h, that allow the finite-difference to be set when evaluating the function. The default values are set for reasonable numerical precision.

Note

WARNING: In the expressions, do not use variable names beginning with a dot, particularly .f or .h

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

Examples

```
g = numD( a*x^2 + x*y ~ x, a=1)
g(x=2,y=10)
gg = numD( a*x^2 + x*y ~ x&x, a=1)
gg(x=2,y=10)
ggg = numD( a*x^2 + x*y ~ x&y, a=1)
ggg(x=2,y=10)
h = numD( g(x=x,y=y,a=a) ~ y, a=1)
h(x=2,y=10)
f = numD( sin(x)~x)
# slice_plot( f(3,.h=hlim)~h, bounds(h=.00000001 :000001)) %>% gf_hline(yintercept = cos(3))
```

PE_fun1 *Potential energy functions used as examples in* MOSAIC Calculus.

Description

These appear in the "Optimization & constraint" chapter.

Usage

```
PE_fun1(x1, y1)

PE_fun2(x1, y1, x2, y2, x3, y3)
```

Arguments

x1, y1, x2, y2, x3, y3	coordinates of the masses.

Picket *Creates a "picket fence" of points for illustrating numerical integration*

Description

Creates a "picket fence" of points for illustrating numerical integration

Usage

```
Picket(
  domain,
  h = 0.1,
  method = c("left", "right", "center", "trapezoid", "gauss")
)
```

Arguments

domain	domain of integration as used in `slice_plot()`, `Integrate()`, and the similar mosaicCalc functions
h	number giving the width between pickets. Could also have been called `dt` or `dx`, and so on.
method	determines the weights for each element in the picket

`Planets` *NASA data on planets*

Description

NASA data on planets

Usage

```
data(Planets)
```

Format

A data frame with 10 rows and these variables:

- `planet` name of the planet
- `mass` in 10^24 kg
- `diameter` in km
- `density` in kg/m^3
- `gravity` in m/s^2
- `escape_velocity` in km/s
- `rotational_period` in hours
- `day_length` in hours
- `distance_from_sun` in 10^6 km
- `perihelion` in 10^6 km
- `aphelion` in 10^6 km
- `orbital_period` in days
- `orbital_velocity` in km/s
- `orbital_inclination` in degrees
- `orbital_eccentricity`
- `obliquity_to_orbit` in degrees
- `mean_temperature` in C
- `surface_pressure` in bars
- `n_moons` a count
- `ring_system` does the planet have a ring system
- `global_magnetic_field` does the planet have a magnetic field

Details

Two of the rows, MOON and PLUTO, are not considered planets. The orbital parameters of the MOON are with respect to EARTH. Others are with respect to the Sun. Negative rotation period denotes a rotation opposite that of most of the other planets.

Source

https://nssdc.gsfc.nasa.gov/planetary/factsheet/

`Planet_solar` *Solar irradiance of the planets*

Description

Brightness of the sun at the various planets

Usage

```
data(Planet_solar)
```

Format

A data frame with 9 rows

- `planet` Name of the planet (or planetoid)
- `distance` distance from the sun in millions of km
- `irradiance` solar brightness (in what units?)

`plotFun` *Plot functions of one and two variables using lattice system*

Description

This is defined in the `mosaic package`: See `plotFun`

`qspliner` *Create a quadratic spline (inefficiently)*

Description

A handmade function to construct quadratic splines.

Usage

```
qspliner(tilde, data, free = 0)
```

Arguments

tilde	A tilde expression of the form y ~ x specifying the output variable (on the LHS) and the input variable (on the RHS). Together with data, these (x,y) pairs will be the knots for the spline.
data	A data frame containing the variables in tilde.
free	A number specifying the slope of the output function at the last knot. Default is 0: that is, flat.

Details

Unless you have a good reason otherwise, you should be using spliner(), which generates cubic splines, rather than qspliner(). qspliner() is intended only for demonstration purposes.

Value

a function suitable for, for instance, graphing or optimizing

Examples

```
Pts <- tibble(x = seq(-4,4, by=.7), y = dnorm(x))
f <- qspliner(y ~ x, data = Pts)
slice_plot(dnorm(x) ~ x, domain(x=-4:4)) %>%
  slice_plot(f(x) ~ x, color= "blue") %>%
  gf_point(y ~ x, data = Pts, color = "orange", size=4, alpha=0.3) %>%
  gf_lims(y= c(NA,.5))
```

---

rfun *Generate a "natural looking" function of one or multiple variables*

Description

This is defined in the mosaic package: See rfun.

RI_tide *Tide levels from the US NOAA*

Description

- RI_tide: For Providence, RI, Minute-by-minute tide data during April 1-5 2010.
- Anchorage_tide: For Anchorage, AK, Every six-minute data during 2018

Usage

```
RI_tide
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 6426 rows and 3 columns.

Details

Variables:

- `level` meters
- `date_time` GMT
- `hour` time in hours after start of April 2010.

Lat/Long: 41.8071N, -71.4012W Request: `https://tidesandcurrents.noaa.gov/api/datagetter?begin_date=20200`
Other product documentation: https://tidesandcurrents.noaa.gov/api/ Google Map: https://www.google.com/maps/place/Prov
71.4012,14z/data=!4m5!3m4!1s0x89e444e0437e735d:0x69df7c4d48b3b627!8m2!3d41.8071!4d-71.4012

`Robot_stations` *Waypoints on the path of a fictitious robot*

Description

This is used in *MOSAIC Calculus* examples about splining.

Usage

```
Robot_stations
```

Format

A data frame with 16 cases, each of which is an x-y location at the indicated time.

- x,y X-Y location in mm
- t time in seconds

Runners *Running times*

Description

A longitudinal record of running times. There are 5977 individual runners included here, each with a unique `id`. Each runner ran the 10-mile race (See `mosaicData::TenMileRace`) on multiple occasions.

Usage

```
Runners
```

Format

Data frame with 24,334 rows

- `year` the occasion on which the runner ran the Cherry Blossom Ten Miler
- `age` the runners age at the time of that occasion
- `gun` the time (in minutes) from the gun to crossing the finish line.
- `net` time between crossing the start line and crossing the finish line. Sometimes this is missing. It's typically smaller than `gun` because it takes time to reach the starting line.
- `sex` as recorded by the race organizers.
- `previous` How many previous occasions did this person run the race.
- `nruns` The total number of runs recorded for this person (including future runs)
- `start_position` A qualitative indication of how close to the front of the pack the person was positioned at the time the gun went off.

Author(s)

Daniel Kaplan

Source

Scraped from the Cherry Blossom web site by patient, unremunerated toil of the author.

simpleYacasIntegrate *Interface to integration using Ryacas*

Description

Interface to integration using Ryacas

Usage

```
simpleYacasIntegrate(tilde, ...)
```

Arguments

tilde	The tilde expression for the integration
...	Bindings for parameters in tilde

simplify_fun *Turn a 1-line function into an inline formula*

Description

Turn a 1-line function into an inline formula

Usage

```
simplify_fun(fun)

inline_expr(ex, old, new, env)

replace_arg_in_expr(ex, old, new)
```

Arguments

fun	a function, such as produced by makeFun().
ex	an expression
old	the name to be replaced as produced by as.name()
new	the name to be substituted in for arg_name
env	the environment for the expression

`slice_plot` *Plot a function of a single variable*

Description

In a slice plot, there is one independent variable. The graph shows the output of the function versus the independent variable. It's called a slice plot to distinguish it from a contour plot, in which the graph has one axis for each independent variable and the output of the function is shown by color and labels.

Usage

```
slice_plot(
  ...,
  npts = 101,
  color = "black",
  alpha = 1,
  label_text = "",
  label_x = 1,
  label_vjust = "top",
  label_color = color,
  label_alpha = alpha,
  singularities = numeric(0)
)
```

Arguments

`...`	Canonical first three argument: `[gg]`, tilde expression, `[domain]` as well as any parameters to be assigned or re-assigned.
`npts`	Integer, number of points at which to evaluate the function.
`color`	Color of curve
`alpha`	Alpha of curve
`label_text`	character string label to place near the graph curve. Default: none.
`label_x`	number between 0 and 1 indicating the horizontal placement of the `label_text`.
`label_vjust`	vertical justification of label. One of "left", "middle", "right", "bottom", "center", "top", "inward", or "outward"
`label_color`	color of label
`label_alpha`	alpha of label
`singularities`	numeric vector of x positions at which to break the graph.
	Additional arguments will be passed to `geom_line()`. Use, e.g. `color="red"`

Value

ggplot2 layers

Examples

```
## Not run:
slice_plot(sin(x) ~ x, domain(x = range(-5, 15)))
f <- makeFun(sin(2*pi*t/P) ~ t)
slice_plot(f(t, P=20) ~ t, domain(t = -5:10), label_text = "Period 20", label_x=0.9)
slice_plot(x^2 ~ x) # Error: no domain specified
slice_plot(cos(x) ~ x, domain(x[-10:10]) # domain will be -10 < x < 10
# see domain

## End(Not run)
```

`smoother` *Create a smoothing function approximating a cloud of points*

Description

This is defined in the `mosaic` package: See `smoother`.

`SSA_2007` *US Mortality table from 2007*

Description

Mortality table from the US Social Security Adminstration issued in 2007.

Usage

```
SSA_2007
```

Format

A data frame with 240 rows. Each row corresponds to one age year from 0 to 120.

- `Age`: The age year. 0 corresponds to birth through one year of age.
- `Sex`: The sex for which the row is relevant.
- `Mortality`: The fraction of people of that age who died in 2007.
- `LifeExpectancy`: The calculated "life expectancy" at that age for that sex.

Details

Life expectancy is a measure constructed from a simulation. Start with 100,000 people at birth. Use the mortality at each age to follow the living through successive ages. The "life expectancy at birth" (age 0) is the average age at death of those 100,000 people. For the life expectancy in year n, consider only that fraction of the original 100,000 who survived to year n. The life expectancy will be the average time until death for those survivors.

`streamlines` *Dynamical systems calculations and graphics*

Description

- `streamlines()` draws raindrop-shaped paths at a randomized grid of points that follow trajectories
- `flow_field()` draws arrows showing the flow at a grid of points
- `trajectory_euler()` compute an Euler solution. ()

Usage

```
streamlines(..., npts = 8, dt = 0.01, nsteps = 10, color = "black", alpha = 1)

flow_field(..., npts = 8, scale = 0.8, color = "black", alpha = 1)

trajectory_euler(..., dt = 0.01, nsteps = 4, full = TRUE, every = 1)
```

Arguments

`...`	The first arguments should describe the dynamics. See details.
`npts`	The number of points on an edge of the grid
`dt`	time step (e.g. 0.01)
`nsteps`	how many Euler steps to take
`color`	What color to use
`alpha`	What alpha to use
`scale`	Number indicating how long to draw arrows. By default, the longest arrows take up a full box in the grid
`full`	report the derivative and the step size for each variable
`every`	n, report will contain every nth step

Details

The dynamical functions themselves will be formulas like `dx ~ a*x*y` and `dy ~ y/x`. Initial conditions will be arguments of the form `x=3` and `y=4`. If there are parameters in the dynamical functions, you should also add the parameter values, for instance `a=2`.

For `flow_field()` and `streamlines()` you do not need to specify initial conditions. The grid will be set by the domain argument.

The graphics functions are all arranged to accept, if given, a ggplot object piped in. The new graphics layer will be drawn on top of that. If there is no ggplot object piped in, then the graphics will be made as a first layer, which can optionally piped into other ggformula functions or `+` into ggplot layers.

Examples

```
streamlines(dx ~ x+y, dy~ x-y, domain(x=0:6, y=0:3))
flow_field(dx ~ x+y, dy~ x-y, domain(x=0:6, y=0:3))
Dyn <- makeODE(dx ~ 0.06*x, dy ~-x - y, domain(t=0:20), dt=0.1,x=3, y=4)
streamlines(Dyn, domain(x=-5:5, y=-5:5), npts=15)
flow_field(Dyn, domain(x=-6:6, y=-10:10))
trajectory_euler(dx ~ -y, dy ~ .5*x, x=3, y=3)
rabbits <- drabbit ~ 0.2*rabbit - 0.01*rabbit*fox
foxes <- dfox ~ -.2*fox + 0.0005*rabbit*fox
trajectory_euler(rabbits, foxes, rabbit=100, fox=3, dt=0.1, nsteps=500, every=10)
```

---

surface_plot *Make an interactive plotly plot of a function of two variables*

Description

An interactive plot lets you interrogate the plot to get numerical values at each point. When `type = "surface"`, the plot can be rotated to see the "shape" of the function from various perspectives. Using the interactive controls, you can save the plot as a PNG file. But it's not possible to overlay plots the way you can with `contourPlot()`.

Usage

```
surface_plot(
  formula,
  domain = c(-5, 5),
  npts = 50,
  type = c("both", "surface", "contour", "heatmap")
)

interactive_plot(
  formula,
  domain = c(-5, 5),
  npts = 50,
  type = c("both", "surface", "contour", "heatmap")
)

surface_with_contours(formula, domain = c(-5, 5), npts = 50)
```

Arguments

`formula`	a formula describing a function in the manner of `mosaicCalc::makeFun()`
`domain`	a call to the `domain()` function giving ranges using the same independent variables as in the `formula`
`npts`	The fineness at which to evaluate the function specified by the formula in the plot. Default: 50
`type`	Plot type: `"surface"`, `"contour"`, `"both"`, or `"heatmap"`

Examples

```
## Not run:
interactive_plot(
    sin(fred*ginger) ~ fred + ginger,
    domain(fred=range(0,pi),
           ginger = range(0, pi)),
    type = "both")

## End(Not run)
```

`symbolicD` *Symbolic Derivatives*

Description

Constructs symbolic derivatives of some mathematical expressions

Usage

```
symbolicD(tilde, ..., .order)
```

Arguments

`tilde`	a tilde expression with the function call on the left side and the w.r.t. variables on the right side.
`...`	additional parameters, typically default values for mathematical parameters
`.order`	a number specifying the order of a derivative with respect to a single variable

Details

Uses the Derivs package for constructing the derivative The `.order` argument is just for convenience when programming high-order derivatives, e.g. the 5th derivative w.r.t. one variable.

When re-assigning default values for arguments in a function being called, as in `D(dnorm(x, mean=3) ~ x)`, you will get a numerical derivative even when the analytic form is known. To avoid this (when possible) use `D(dnorm(x) ~ x, mean=3)`

Value

a function implementing the derivative

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

See Also

`D`, `numD`, `makeFun`, `antiD`, `plotFun`

Examples

```
symbolicD( a*x^2 ~ x)
symbolicD( a*x^2 ~ x&x)
symbolicD( a*sin(x)~x, .order=4)
symbolicD( a*x^2*y+b*y ~ x, a=10, b=100 )
symbolicD( dnorm(x, mn, sd) ~ x, mn=3, sd=2)
```

---

traj_plot *Plots a trajectory*

Description

This function handles trajectories can stem from either of two sources:

1. A parametric description of a curve, such as `sin(t) ~ cos(t)`, along with a domain in t.
2. The solution to an ordinary differential equation as produce by `integrateODE()`

Usage

```
traj_plot(..., npts = 500, nt = 5)
```

Arguments

`...`	Handles the first several objects which are, in this order - tilde: a two sided tilde expression - soln: optionally, a solution object such as from `integrateODE()`, or instead - domain: a domain object, e.g. `domain(t=0:10)`
`npts`	number of plotted points (default: 500)
`nt`	number of tick marks to use in a trajectory plot

Details

The tilde expression is the critical part

Examples

```
traj_plot(2*x + 3 ~ sin(x), domain(x=0:10))
PPdyn <- makeODE(dR ~  0.3*R - 0.03*R*F, dF ~ -0.3*F + 0.0003*R*F)
Soln <- integrateODE(PPdyn, domain(t=0:20), R=1200, F=8)
traj_plot(R(t) ~ F(t), Soln, nt=10)
traj_plot(R(t)*F(t) ~ t, Soln, nt=0)
```

traj_plot_3D *Simple 3D plot of a trajectory*

Description

Takes a trajectory with three state variables as produced by `integrateODE()` and plots out in a 3-dimensional perspective plot, which can be rotated.

Usage

```
traj_plot_3D(x, y, z, soln, domain = NULL, npts = 1000)
```

Arguments

`x`	Name of one of the state variables to be plotted.
`y`	Similar to `x`
`z`	Similar to `y` and `x`
`soln`	Solution output from `integrateODE()`
`domain`	Optional list like `domain=domain(t=c(0,100))`. By default, this will be inferred from `soln`
`npts`	Number of points at which to evaluate the solution.

Examples

```
Lorenz <- makeODE(dx ~ sigma*(y-x), dy ~(x*(rho-z) - y), dz ~ (x*y - beta*z),
                  rho=28, sigma=10, beta = 8/3)
T1 <- integrateODE(Lorenz, domain(t=0:50), x=-5, y=-7, z=19.4)
traj_plot_3D(x, y, z, T1, npts=5000)
```

UK_GDP *Gross Domestic Product of the United Kingdom over a millenium*

Description

An estimate of the real GDP of the United Kingdom over about 7 centuries. "Real" means adjusted for inflation. Take such estimates with a large amount of salt, since pre-industrial era data is very limited and because the components of GDP change substantially over a few decades, let alone a few centuries. Also, the name `UK_GDP` is anachronistic, since for most of the record is before the UK was created as a political entity.

Usage

```
data(UK_GDP)
```

Format

A data frame with 747 rows

- `year` Which year the estimate is for.
- `GDP` *per capita* Gross domestic product in GBP (pounds)

`unbound` *Identifying unbound inputs to a function*

Description

`unbound()` finds if there are any unbound parameters in a function. This can be useful for checking before handing a function over to a numerical routine. `bind_params()` lets you add parameter bindings or override existing ones.

Usage

```
unbound(f)

bind_params(f, ...)
```

Arguments

`f`	a function (not a tilde expr.)
`...`	bindings for parameters

`US_income` *Income distribution data from the US in 2009*

Description

Data such as these are used to study income inequality. One measure of this is the "Gini coefficient," which can be estimated from a Lorenz function fitted to such data.

Usage

```
US_income
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 6 rows and 2 columns.

Source

La Haye and Zizler (2021) "The Lorenz Curve in the Classroom", *The American Statistician*, 75(2):217-225]

Variables:

- `income` The fraction of total household income that goes to the corresponding poorest fraction of the population.
- `pop` The fraction of the population whose aggregated income is reported as the `income` value.

`vectors` *Utilities for vector calculations*

Description

`%dot%`, `%onto%`, and `%perp%` are infix operators. The left-hand argument is a vector.

Usage

```
u %dot% b

b %onto% A

b %perp% A

normalize(A)

as_magnitude(A, metric = c("2", "O", "I", "F", "M"))
```

Arguments

`u`	a row vector, but a column vector is acceptable too
`b`	a column vector
`A`	a matrix
`metric`	metric to use for matrix norm

Details

Convenience functions for basic operations relating to vector projection. These use the *MOSAIC Calc* conventions that require vectors to be one column matrices.

Value

either a number (for `%dot%`) or a vector

`vector_arg` *convert a function with separate arguments into one with a single vector argument For use with optim.*

Description

convert a function with separate arguments into one with a single vector argument For use with optim.

Usage

```
vector_arg(f)
```

Arguments

`f`	a function with multiple arguments

`Vowel_ee` *Recordings of vowel sounds*

Description

"Oh" refers to "o" as in "stone." "Ee" to "eel".

Usage

```
Vowel_ee

Vowel_oh

Oh_sound

Ee_sound
```

Format

Data frames with two columns

- `y`: The instantaneous amplitude of the vowel sound.
- `t`: Time (seconds)

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 780 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7192 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 10194 rows and 2 columns.

Details

- Oh_sound and Ee_sound contain the complete utterances.
- Vowel_ee and Vowel_oh are snippets of about 0.04 seconds duration.

Zeros *Finds zeros of a function within a specified domain*

Description

Finds zeros of a function within a specified domain

Usage

```
Zeros(tilde, domain = NULL, ..., nsegs = 131)
```

Arguments

tilde	tilde expression defining a function, suitable for makeFun()
domain	specification of domain, as in slice_plot()
...	Assignments to parameters
nsegs	Subdivide the domain into this many segments, looking for a zero in each of those segments. This helps to find multiple zeros.

Value

A data frame with two columns. The first has the name of the input in the tilde expression, and gives the values for that input at which the function is approximately zero. The second column, .output. gives the actual value of the function at the inputs in the first column.

Examples

```
Zeros(a*x + b ~ x, a=1, b=2)
```

Index