Package: fieldtrial (via r-universe)

November 14, 2026

Title Parameter Sets, Trait Trade-Offs and Environmental Drivers for the 'plant' Model

Version 0.0.0.9000

Description Sets up runs of the plant forest model by bundling a site's environment with a species' trait trade-offs and parameter sets. Holds named parameter sets for species and sites, the functions that derive model parameters from traits, and adapters that turn gridded and flux-tower climate into the daily driver series plant reads.

License AGPL (>= 3)

URL https://github.com/traitecoevo/fieldtrial

BugReports https://github.com/traitecoevo/fieldtrial/issues

Depends R (>= 4.5.0)

Imports plant, stats, yaml

Suggests testthat (>= 3.0.0), withr, geodata, terra, ncdf4

Remotes traitecoevo/plant@83a973ffe771e841de46cd5ecb1575b6f8f16f84

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-11-14 02:37:10 UTC

RemoteUrl https://github.com/traitecoevo/fieldtrial

RemoteRef HEAD

RemoteSha 4eba1976f336d440ee142a4b726b2033f2500ebc

Contents

apply_drivers . . . 2
check_passthrough . . . 4
met_climatology . . . 4
met_daily_from_subdaily . . . 5
met_from_monthly . . . 7
met_from_worldclim . . . 8
met_schema . . . 10
met_time . . . 10
parameter_set . . . 11
pulse_series_fit . . . 12
pulse_series_poisson . . . 14
soil_widths_graded . . . 15
solar_declination . . . 17
swdown_clear_sky . . . 18
swdown_noon_from_mean . . . 19
tradeoff_coefficients . . . 19
tradeoffs . . . 20

Index **22**

`apply_drivers` *Set a plant environment's drivers from a daily met table*

Description

Writes the daily met table's columns into a TF24 environment's drivers, one knot per day at its middle, and turns its precipitation into rainfall:

Usage

```
apply_drivers(
  env,
  met,
  start_year = NULL,
  rainfall = c("pulse", "spline"),
  drivers = c("SWdown", "atm_vpd", "atm_vpd_diurnal_amp", "atm_kpa", "wind_speed", "ca",
    "rainfall"),
  until = NULL
)
```

Arguments

`env`	A TF24-family `plant::Environment`, changed in place.
`met`	A daily met table (see `met_schema()`).
`start_year`	The calendar year at time 0. Defaults to the table's first year.
`rainfall`	`"pulse"` or `"spline"`.

`drivers` The drivers to set, from those above; `"rainfall"` stands for precipitation. Defaults to all but `leaf_temp`. A driver whose column is absent is left at plant's value and reported.

`until` Drop rainfall pulses after this time: plant refuses an event past the run's `max_patch_lifetime`, so pass that. `NULL` keeps them all.

Details

met column	driver	conversion
`SWdown_noon`	`SWdown`	none, W m-2
`VPD_noon`	`atm_vpd`	none, kPa
`VPD_amp`	`atm_vpd_diurnal_amp`	only where the environment has it
`Psurf`	`atm_kpa`	none, kPa
`Wind`	`wind_speed`	none, m s-1
`CO2air`	`ca`	ppm to Pa at `Psurf`, or at the environment's `atm_kpa`
`Tair_noon`	`leaf_temp`	only if `"leaf_temp"` is in `drivers`
`Precip`	rainfall	see `rainfall`

`leaf_temp` is left at plant's value unless asked for, because plant's TF24 runs its leaf at that temperature with the energy balance off, and a site's air temperature changes every temperature response at once. When asked for, it is the noon temperature, like every other driver here: the daily mean would pair a noon VPD with a cooler, less saturable air.

Rainfall is delivered as **pulses** by default: each wet day's total, in metres, at the middle of the day, as rows from `plant::rainfall_pulse()`, and the continuous `rainfall` driver is set to zero so the water is not counted twice. Pass the returned `events` to `plant::run()` with `plant::events()`. A daily total is a sum of discrete events; interpolating it spreads each one over its neighbours. `rainfall = "spline"` sets the continuous driver instead (m yr-1), which suits a mean year.

Before setting anything, the noon VPD is checked against the leaf's limit, `0 < VPD < e_s(T)` at the leaf temperature it will meet (Tetens, as plant's leaf computes it): plant refuses outside it, and it is cheaper to say so here, with the dates.

Times are plant's: years from 1 January of `start_year`. plant's TF24 reads the day of year from the fractional part of time, so `start_year` must be the first calendar year of the run. Each day's value is a knot at the middle of the day, and the first and last days hold to the start of the first and the end of the last, so the drivers cover the table's whole days; a run must not go past them.

Value

A list: `env`; `events`, the rainfall pulses (or `NULL`); and `drivers`, a data frame saying, for each driver, whether it was set from the table or left at plant's value.

check_passthrough *Which leaf parameters plant can set*

Description

plant's TF24 family hands its leaf (phylloptim's `Leaf`) a set of traits and numerical controls. A parameter set can only reach what plant exposes: a strategy parameter, a strategy field, or a `Control` field. This lists every trait in `phylloptim::leaf_traits()` and every control in `phylloptim::leaf_control()` with where plant exposes it, so a new leaf parameter that plant does not yet forward shows up as a gap to fix in plant, not as a value a set silently cannot change.

Usage

```
check_passthrough(model = "TF24")
```

Arguments

`model`	A TF24-family model name.

Details

Two kinds of leaf parameter are listed as out of scope rather than gaps: those of the leaf's other stomatal models (`JS22_gamma`, `CMax_a`, `CMax_b`), which plant's TF24 does not select, and `integration_tol`, which only sets the tolerance of a direct quadrature off the production path that plant never calls. A plant field for either would be settable and reach nothing.

Value

A data frame with columns `name`, `kind` (`"trait"` or `"control"`) and `where` (`"strategy pars"`, `"strategy field"`, `"Control"`, `"out of scope"`, or `NA` for a gap).

met_climatology *Mean year and repeated years*

Description

`met_climatology()` collapses a record to its mean year: each column averaged by day of year over the complete years, with 29 February dropped, dated in the non-leap year 2001. `met_repeat()` repeats whole years of a table end to end to cover a longer run, since plant's drivers cannot be evaluated outside the times they were given.

Usage

```
met_climatology(met)

met_repeat(met, n_years)
```

Arguments

`met`	A daily met table.
`n_years`	How many years the result spans.

Details

A mean year's rainfall falls a little on every day. Deliver it as a spline (`apply_drivers(rainfall = "spline")`), or draw pulses from the record with `pulse_series_fit()` instead.

Value

A daily met table carrying `met`'s site attributes (`latitude`, `longitude`, `vpd_peak_hour`, `filled`, `source`, `climatology_of`). `met_climatology()` sets `climatology_of` to the first and last years averaged.

Examples

```
met <- data.frame(date = seq(as.Date("2001-01-01"), as.Date("2003-12-31"), by = "day"))
met$Tair_mean <- 20 + 8 * cos(2 * pi * seq_len(nrow(met)) / 365)
nrow(met_climatology(met))
nrow(met_repeat(met_climatology(met), 10))
```

`met_daily_from_subdaily` *Daily met table from half-hourly or hourly records*

Description

Aggregates a tower or reanalysis record to the daily met table, fitting the noon values and VPD amplitude plant's TF24 rebuilds a day from:

Usage

```
met_daily_from_subdaily(
  x,
  latitude,
  longitude,
  utc_offset,
  timestamp = c("start", "middle", "end"),
  vpd_peak_hour = 14.5,
  max_amp = 0.99,
  vpd_floor = 0.01
)
```

Arguments

`x`	A data frame with a `time` column (`POSIXct`, or anything `as.POSIXct()` reads, in local standard time) and any of `SWdown` (W m-2), `Tair` (degC), `VPD` (kPa), `Precip` (mm per step), `Wind` (m s-1), `Psurf` (kPa) and `CO2air` (ppm).
`latitude`, `longitude`	Site location, degrees; south and west negative.
`utc_offset`	Hours the timestamps are ahead of UTC (Alice Springs standard time: 9.5). Daylight saving is not followed.
`timestamp`	Whether `time` marks the `"start"`, `"middle"` or `"end"` of each step. FLUXNET's `TIMESTAMP_START` is the start.
`vpd_peak_hour`	Local solar hour of the daily VPD peak.
`max_amp`, `vpd_floor`	Bounds applied to the VPD amplitude and the noon VPD (kPa).

Details

- `SWdown_noon` is the noon value of a clear-sky-shaped day holding the day's observed shortwave energy (`swdown_noon_from_mean()`), so the day plant rebuilds integrates to what was measured.
- `VPD_amp` and `VPD_noon` come from a least-squares fit of `VPD(h) = c0 * (1 + A * cos(2 * pi * (h - vpd_peak_hour) / 24))` to the day's record, in local solar time.
- `Tair_noon` comes from the same fit applied to air temperature (with no amplitude bound); the day's mean, maximum and minimum are kept too.
- Precipitation is the day's total; wind and CO2 their means while the sun is up; pressure its mean.

Records are put into days by local solar time, worked out from the timestamps' fixed offset from UTC, the longitude and the equation of time. A day is kept only if it is complete (a solar day may hold one step more or fewer than 24 clock hours), so the partial days at each end of a record are dropped.

Two corrections keep the table inside what plant accepts, and are counted in the result's attributes: an amplitude outside `[0, max_amp]` (a day whose VPD peaks at night, say) is clamped, and a noon VPD below `vpd_floor` is raised to it, since the leaf refuses a saturated atmosphere.

Value

A daily met table (see `met_schema()`) with attributes `latitude`, `longitude`, `vpd_peak_hour`, `filled` (empty: every column is from the record), `amp_clamped` and `vpd_floored` (the days corrected).

`met_from_monthly` *Daily met table from monthly climate*

Description

Turns monthly climate, either normals (one row per month) or a series (one row per year and month), into the daily met table. Every monthly source in fieldtrial (WorldClim, and later CRU TS and AGCD) goes through this, so they share one set of assumptions:

Usage

```
met_from_monthly(
  monthly,
  latitude,
  longitude = NA_real_,
  elevation = NULL,
  peak_hour = 14.5,
  max_amp = 0.99,
  vpd_floor = 0.01,
  source = NULL
)
```

Arguments

`monthly`	A data frame with `month` (1 to 12), optionally `year`, and any of `Tair_mean`, `Tair_max`, `Tair_min` (degC), `Precip_month` (mm per month), `SWdown_mean` (W m-2, 24-hour mean), `Wind` (m s-1, 24-hour mean) and `vapour_pressure` (kPa). Without `year`, it must be the 12 months of a climatology; with it, consecutive months.
`latitude`, `longitude`	Site location, degrees.
`elevation`	Site elevation, m, for `Psurf`; `NULL` leaves `Psurf` out.
`peak_hour`	Local solar hour of the daily temperature and VPD peak.
`max_amp`, `vpd_floor`	Bounds on the VPD amplitude and noon VPD (kPa), as in `met_daily_from_subdaily()`.
`source`	Recorded as `attr(, "source")`.

Details

- **Daily values** are interpolated linearly between mid-month values, wrapping December to January for normals. Precipitation and shortwave are then rescaled month by month so each month keeps its total exactly; temperatures, vapour pressure and wind keep their mid-month values but not exactly their monthly means.
- **The day's temperature** follows one cosine between `Tair_min` and `Tair_max`, peaking at `peak_hour`; `Tair_noon` is read from it.

- **VPD** through the day is `e_s(T(h)) - e_a`, with the actual vapour pressure `e_a` constant over the day; `VPD_noon` and `VPD_amp` are the least-squares cosine fit to it that tower records get, bounded the same way.
- **Noon shortwave** is the noon value of a clear-sky-shaped day holding the daily mean (`swdown_noon_from_mean()`).
- **Pressure** is from `elevation` by the standard atmosphere, if given.

The columns estimated by an assumption rather than read from the source are named in `attr(, "filled")`.

Value

A daily met table (see `met_schema()`). Normals are dated in the non-leap year 2001 and carry `climatology_of` if `monthly` has that attribute.

Examples

```
normals <- data.frame(month = 1:12,
  Tair_min = c(22, 21, 18, 13, 8, 5, 4, 6, 10, 15, 18, 21),
  Tair_max = c(36, 35, 32, 28, 23, 20, 19, 22, 27, 31, 34, 35),
  Precip_month = c(40, 45, 30, 15, 15, 12, 8, 7, 8, 20, 25, 40),
  SWdown_mean = c(310, 290, 255, 210, 170, 150, 165, 200, 245, 280, 300, 315),
  vapour_pressure = c(1.4, 1.5, 1.3, 1.0, 0.9, 0.8, 0.7, 0.7, 0.7, 0.9, 1.0, 1.2))
normals$Tair_mean <- (normals$Tair_min + normals$Tair_max) / 2
met <- met_from_monthly(normals, latitude = -22.28, longitude = 133.25, elevation = 600)
head(met)
```

`met_from_worldclim` *Daily met table from WorldClim*

Description

WorldClim 2.1 gives monthly climate normals for 1970-2000 on a global grid (Fick and Hijmans 2017). `met_from_worldclim()` reads them at a site and makes a daily mean year with `met_from_monthly()`; `worldclim_monthly()` returns the monthly values themselves.

Usage

```
met_from_worldclim(
  latitude,
  longitude,
  res = 10,
  path = fieldtrial_cache("worldclim"),
  ...
)

worldclim_monthly(
  latitude,
```

```
  longitude,
  res = 10,
  path = fieldtrial_cache("worldclim")
)
```

Arguments

`latitude`, `longitude`	Site location, degrees.
`res`	Grid resolution in arc-minutes: 10, 5, 2.5 or 0.5.
`path`	Folder to keep the downloads in.
`...`	Passed to `met_from_monthly()`.

Details

The grids are downloaded once with the `geodata` package and kept in `path`. A download is the whole globe for each of seven variables plus elevation, so the resolution sets its size: about 200 MB at 10 arc-minutes (the default, about 18 km), 700 MB at 5, 2.5 GB at 2.5. At 0.5 (30 arc-seconds, about 1 km) the 30-degree tile around the site is fetched instead.

What WorldClim gives and what is assumed: temperatures, precipitation, shortwave (`srad`, converted from kJ m-2 d-1 to W m-2), wind (a 24-hour mean, used where the table wants a daytime mean) and vapour pressure are read; the noon temperature, noon VPD and its amplitude come from the daily cosine assumptions in `met_from_monthly()`; surface pressure comes from elevation. There is no CO2. The result is a mean year with no interannual variation, so its rainfall belongs on a spline (`apply_drivers(rainfall = "spline")`).

Value

`met_from_worldclim()`: a daily met table for the year 2001, with `climatology_of = c(1970, 2000)`. `worldclim_monthly()`: a data frame of 12 months in `met_from_monthly()`'s columns, with an `elevation` attribute.

References

Fick, S.E. and Hijmans, R.J. (2017) WorldClim 2: new 1-km spatial resolution climate surfaces for global land areas. International Journal of Climatology 37(12), 4302-4315.

Examples

```
## Not run:
met <- met_from_worldclim(-22.283, 133.249)  # Alice Springs Mulga
sum(met$Precip)                               # mean annual rainfall, mm

## End(Not run)
```

met_schema *The daily met table*

Description

Every climate source in fieldtrial produces the same daily table, and `apply_drivers()` turns it into plant's drivers. `met_schema()` describes its columns; `validate_met()` checks a table against it and returns it, or stops saying what is wrong.

Usage

```
met_schema()

validate_met(met)
```

Arguments

`met`	A data frame.

Details

The table holds **noon values**, not daytime means: plant's TF24 reads `SWdown` and `atm_vpd` as their values at solar noon and rebuilds the day from them, with a VPD amplitude and the latitude. `Tair_noon` is the air temperature the same noon, and the one a noon VPD must be read against. Only `date` is required; a source supplies the columns it can. Site metadata travel as attributes: `latitude`, `longitude` and `vpd_peak_hour` (local solar time of the VPD peak), and `filled`, naming the columns that were estimated rather than observed.

Value

`met_schema()`: a data frame of columns, units, bounds and meanings. `validate_met()`: `met`, invisibly.

Examples

```
met_schema()
```

met_time *Time in plant's convention for calendar dates*

Description

Years from 1 January of `start_year`, placed at the middle of each day. plant's TF24 reads the day of year as `365 * (time - floor(time))`, so `start_year` must be the run's first calendar year for the seasons to line up.

Usage

```
met_time(date, start_year)
```

Arguments

`date`	Dates, as `Date`.
`start_year`	The calendar year at time 0.

Value

Numeric times, in years.

Examples

```
met_time(as.Date(c("2010-01-01", "2010-07-02", "2011-01-01")), 2010)
```

`parameter_set` *Parameter sets for species and sites*

Description

A parameter set bundles everything one run of plant needs for a species at a site: its traits and the trade-offs that derive its other parameters, any parameters set directly, the soil, the climate drivers and the run's settings, each scientific value with the source it came from. Sets live as YAML files in the package (`list_sets()`) or anywhere on disk.

Usage

```
parameter_set(
  set,
  check = TRUE,
  drivers = TRUE,
  driver_path = fieldtrial_cache("worldclim")
)

check_set(set, allow_version_drift = FALSE)

run_set(set, ...)

list_sets()
```

Arguments

`set`	A set name from `list_sets()`, a path to a set file, or a `fieldtrial_set`.
`check`	Whether to run `check_set()` on the result.

`drivers`	Whether to build the set's drivers. `FALSE` builds everything else, leaving the environment's drivers at plant's values and `events` empty, so a set can be checked without its climate data.
`driver_path`	Where downloaded climate data are kept, for sets whose drivers come from a gridded source.
`allow_version_drift`	Accept a set built against another plant model version.
`...`	Passed to `plant::run()`.

Details

`parameter_set()` reads a set and builds the four inputs of `plant::run()`: `p` (a `plant::Parameters` with one strategy), `env` (a `plant::Environment` with its soil and drivers), `ctrl` (a `plant::Control`) and `events` (rainfall pulses, or `NULL`). `run_set()` runs them. Nothing a set names is dropped: a key that reaches none of these inputs is refused.

A set records the plant model version it was built against (`plant_scientific_version`, e.g. `"TF24@v15"`). `check_set()` compares it with the installed plant and refuses a mismatch, since the same numbers can mean a different plant after a scientific change.

Value

`parameter_set()`: a list of class `fieldtrial_set` with `p`, `env`, `ctrl`, `events`, `tradeoffs` (the trade-off function used), `met` (the daily met table) and `meta` (the parsed file). `run_set()`: the `plant_run` that `plant::run()` returns. `check_set()`: `set`, invisibly. `list_sets()`: the names of the sets in the package.

`pulse_series_fit` *Series of resource pulses fitted to a daily record*

Description

Series of resource pulses fitted to a daily record

Usage

```
pulse_series_fit(date, amount, wet_threshold = 0, min_wet = 30)

pulse_series_markov(from, to, fit, days_per_pulse = 1)
```

Arguments

`date`	Dates of the record's days, as `Date` or something `as.Date()` takes.
`amount`	Amount on each day, `NA` where the day is missing, in the units the pulse constructor takes (TF24: metres of water).
`wet_threshold`	A day is wet when its amount is above this.
`min_wet`	The fewest days a month and class needs for a fit of its own.

`from`, `to` The span the series covers: pulses fall in `(from, to]`.

`fit` A fit from `pulse_series_fit()`.

`days_per_pulse` Days whose amounts make one pulse: 1 for a pulse each wet day, 7 for one each week.

Details

`pulse_series_fit()` fits a seasonal weather generator to an observed daily record, and `pulse_series_markov()` draws a series of any length from the fit. Where `pulse_series_record()` repeats a record, which makes the series periodic, a fitted series is stationary, and a different realisation for each seed.

The fit uses the record's complete years only: those with an observation on every day. Each year is put in a class by its total – dry, normal or wet, split at the terciles of the annual totals. For each month and class, the fit keeps the chance that a day is wet after a wet day and after a dry one (a first-order Markov chain), and the empirical quantiles of the amounts on wet days. A month and class with fewer than `min_wet` wet days, or fewer than `min_wet` days after a wet or a dry day, takes that month's values pooled over the classes.

`pulse_series_markov()` draws each year in turn: its total from the quantiles of the annual totals, and so its class; the wet days from that class's chain, month by month; and each wet day's amount from that month's quantiles. It then scales the year's amounts to the drawn total. A model year has 365 days and starts on 1 January, at `from`. Each day's amount is a pulse at the middle of its day, or, with `days_per_pulse` above 1, the amounts of that many days are one pulse at the middle of the days. A pulse of zero is left out. The draws come from R's random number stream, so `set.seed()` fixes the series.

Value

`pulse_series_fit()`: a list with the quantile levels `probs`, the quantiles of the annual totals `annual`, their `terciles`, the 12 x 3 (month by class) matrices `p_ww` and `p_wd` of the chance of a wet day after a wet and a dry day, the 12 x 3 x `length(probs)` array `depth` of quantiles of the amounts on wet days, and the complete `years` used. `pulse_series_markov()`: a data frame with columns `time` and `amount`.

See Also

`pulse_series_poisson`, `rainfall_pulse`

Examples

```
set.seed(1)
date <- seq(as.Date("1991-01-01"), as.Date("2020-12-31"), by = "day")
wet <- runif(length(date)) < ifelse(format(date, "%m") %in% c("12", "01", "02"), 0.5, 0.1)
amount <- ifelse(wet, rexp(length(date), rate = 1 / 0.01), 0)
fit <- pulse_series_fit(date, amount)
s <- pulse_series_markov(0, 100, fit, days_per_pulse = 7)
sum(s$amount) / 100          # about the record's mean annual total
```

`pulse_series_poisson` *Series of resource pulses*

Description

Series of resource pulses

Usage

```
pulse_series_poisson(from, to, rate, mean_amount)

pulse_series_regular(from, to, interval, amount)

pulse_series_record(
  time,
  amount,
  from = 0,
  to = from + period,
  period = ceiling(max(time))
)
```

Arguments

`from`, `to`	The span the series covers: pulses fall in `(from, to]`.
`rate`	Mean number of pulses per year.
`mean_amount`	Mean amount per pulse, in the units the pulse constructor takes (TF24: metres of water).
`interval`	Years between pulses.
`amount`	Amount per pulse (`pulse_series_regular`), or per recorded pulse (`pulse_series_record`).
`time`	Times of the recorded pulses, in `(0, period]` and increasing: day of record / 365.25, say.
`period`	The record's length in years, the spacing at which it repeats. Defaults to its last time rounded up to a whole year.

Details

Three ways to build a series of pulses for a run, each returning the pulses' times and amounts as a data frame, sorted by time, to hand to a model's pulse constructor: for TF24, `rainfall_pulse(s$time, s$amount)`.

`pulse_series_poisson()` draws a stationary marked Poisson process: pulses arrive at `rate` per year, and each one's amount is exponential with mean `mean_amount`, so the long-run supply is `rate * mean_amount` per year. It draws from R's random number stream, so `set.seed()` fixes the series, and one realisation shared by every run that is compared is what makes a comparison of fitness a comparison of traits (common random numbers). It

draws the count, then the times, then the amounts, so a series over a longer span does not contain a shorter one.

`pulse_series_regular()` puts a pulse of `amount` every `interval` years.

`pulse_series_record()` takes an observed record – each wet day a pulse, say, with times in years – and, given `to` beyond the record's end, repeats it end to end to cover a longer run: a 100-year record behind a burn-in and a window of several centuries. Repeating a record makes the series periodic, which a window shorter than the record does not see.

Value

A data frame with columns `time` and `amount`.

See Also

`rainfall_pulse`, `events`

Examples

```
set.seed(1)
s <- pulse_series_poisson(0, 100, rate = 20, mean_amount = 0.014)
nrow(s)                       # about 2000 pulses
sum(s$amount) / 100           # about 0.28 m of water a year
p <- plant::add_strategies(plant::base_parameters("TF24"),
                           plant::trait_matrix(0.0825, "lma"))
ev <- plant::events(plant::events_default(p), plant::rainfall_pulse(s$time, s$amount))
```

`soil_widths_graded` *Soil layer widths grading from a thin surface layer*

Description

Builds a vector of layer widths for `set_soil_layer_widths()` on a TF24 environment in plant: `n` layers spanning `depth`, the first of width `top` and each subsequent one a fixed factor thicker. The motivating case is soil evaporation, which needs a very thin layer at the top of the profile – a 30 cm surface layer averages away the wetting and drying that drives it.

Usage

```
soil_widths_graded(depth = 1.5, n = 5, top = 0.02)
```

Arguments

`depth`	Total column depth, m.
`n`	Number of layers.
`top`	Width of the surface layer, m. Must be less than `depth / n`, since a graded profile thickens with depth.

Details

The ratio is solved so the widths sum to `depth` exactly; the last width is then taken as the remainder rather than computed, so rounding cannot leave the column a hair shallower or deeper than asked for.

Value

A vector of `n` layer widths, summing to `depth`, each at least as thick as the one above it.

What a thin layer costs

Two things, neither of which `plant` corrects for you.

The column gets stiff. The soil water balance's Jacobian diagonal is `(dK/dtheta)/dz`, so it scales as `1/dz`: measured 1.8e4 yr^-1 at the 0.3 m default and 2.6e5 at 2 cm, where the characteristic time is about two minutes. It only bites when the layer is wet – just after rain, which is exactly when an evaporation layer matters. Below about 2.6 mm the stable step falls under `Control()`'s `ode_step_size_min` and the step controller cannot satisfy stability at all.

Runoff is recalibrated. The infiltration term keys off the surface layer alone, through `(theta_0/theta_sat)^b_infil` with `b_infil = 8`. A thin layer 0 fills far faster and so holds that switch shut for longer: replacing the uniform profile with a 2 cm top layer moved annual runoff from 13.4\ changed. `a_infil` and `b_infil` were calibrated against a 30 cm surface layer.

Layers may not get thinner with depth

plant's environment refuses, in `set_soil_layer_widths()`, a profile in which any layer is thinner than the one above it (equal widths are fine, so a uniform profile is legal). Refining towards the surface is the standard and is what this function builds; refining at depth has no counterpart in TF24 – a contrast in soil properties is per-layer *parameters*, not a thin layer, and TF24 has no water table – while it does reach a failure the model cannot absorb, driving the size-density equations to overflow. Every profile this function returns satisfies the rule by construction.

A thickening profile is not automatically well resolved either. The error in the usual Richards discretisation vanishes on a uniform grid and grows with the *contrast* between adjacent thicknesses whichever way it runs, and grading fast with depth is equivalent to a coarser column. So a small `top` at a fixed `n` buys surface resolution at the price of a coarse base: at `top = 0.005` the deepest of five layers is 1.11 m. Prefer more layers to a steeper ladder.

See Also

`set_tf24_soil`

Examples

```
soil_widths_graded(depth = 1.5, n = 5, top = 0.02)
sum(soil_widths_graded(depth = 1.5, n = 5, top = 0.02)) # 1.5
```

`solar_declination` *Solar geometry of a day*

Description

`solar_declination()` gives the sun's declination, and `solar_day()` the terms of the sun's elevation through a day at a latitude: sin(elevation) at solar hour `h` is `a + b * cos(pi * (h - 12) / 12)`, and the photoperiod is where that is positive. These are the forms plant's TF24 uses when it rebuilds a day from noon values, so drivers made here and the day plant integrates share one sun.

Usage

```
solar_declination(time)

solar_day(time, latitude)
```

Arguments

`time`	Time in years from 1 January, as plant counts it: the day of year is `365 * (time - floor(time))`.
`latitude`	Latitude in degrees, south negative.

Details

The declination is `asin(0.39785 * sin(...))`. plant's older R solar model (kept for FF16's trade-offs) uses the inner expression as the angle itself, which is about 0.6 degrees low at the solstices.

Value

`solar_declination()`: radians. `solar_day()`: a data frame with columns `a`, `b` and `daylength` (hours), one row per `time`. A day with no sunrise or no sunset (polar day or night) is refused.

Examples

```
solar_day(c(0, 0.5), latitude = -22.28)  # midsummer and midwinter at Alice Springs
```

`swdown_clear_sky` *Clear-sky shortwave*

Description

Shortwave on a horizontal surface under a clear sky at a solar hour, for a location- and season-dependent light driver where radiation was not observed (traitecoevo/plant#701). The beam is `S0 * transmissivity^m` at air mass `m = 1 / sin(elevation)`, projected onto the horizontal.

Usage

```
swdown_clear_sky(time, latitude, hour = 12, transmissivity = 0.75, S0 = 1360)
```

Arguments

`time`	Time in years from 1 January, as plant counts it: the day of year is `365 * (time - floor(time))`.
`latitude`	Latitude in degrees, south negative.
`hour`	Local solar hour; 12 is noon.
`transmissivity`	Fraction of the beam that crosses one air mass, about 0.7 to 0.8 under clear skies.
`S0`	Shortwave outside the atmosphere, W m-2.

Details

This is an estimate of an upper bound, not of a real day: it has no clouds, no diffuse light and no aerosol. Where a gridded product gives daily radiation, use that through `swdown_noon_from_mean()` instead.

Value

W m-2, zero when the sun is down.

Examples

```
t <- (seq_len(365) - 0.5) / 365
plot(t, swdown_clear_sky(t, latitude = -22.28), type = "l")
```

swdown_noon_from_mean

Shortwave at solar noon from a daily mean, and back

Description

A day's shortwave follows the sun's elevation, scaled so that its noon value is `SWdown_noon`; plant's TF24 rebuilds the day this way. These convert between that noon value and the 24-hour mean the same day carries, so a day rebuilt from `SWdown_noon` holds exactly the observed energy.

Usage

```
swdown_noon_from_mean(SWdown_mean, time, latitude)

swdown_mean_from_noon(SWdown_noon, time, latitude)
```

Arguments

`SWdown_mean`	Shortwave averaged over 24 hours, W m-2.
`time`	Time in years from 1 January, as plant counts it: the day of year is `365 * (time - floor(time))`.
`latitude`	Latitude in degrees, south negative.
`SWdown_noon`	Shortwave at solar noon, W m-2.

Value

The other quantity, W m-2.

Examples

```
swdown_noon_from_mean(250, time = 0.05, latitude = -22.28)
```

tradeoff_coefficients

Trade-off coefficients for a model

Description

The coefficients that parameterise a model's trait trade-offs, read from the package's coefficient file for that model's family. TF24, TF24f, TF24t and TF24w share one set.

Usage

```
tradeoff_coefficients(model)

tradeoff_table(model)
```

Arguments

`model`	A plant model name, such as `"TF24"` or `"FF16"`.

Value

`tradeoff_coefficients()`: a named list of numbers. `tradeoff_table()`: a data frame with one row per coefficient and columns `name`, `value`, `units`, `description`, `source` and `reference`.

Examples

```
tradeoff_coefficients("TF24")$B_Hv1
head(tradeoff_table("FF16"))
```

---

tradeoffs *Trait trade-offs for a plant model*

Description

Builds the function plant uses to derive model parameters from traits: it takes a trait matrix `m` (one row per strategy, named columns) and a strategy `s`, and returns `m` with the derived parameters appended. Pass it to `plant::add_strategies()`, `plant::generate_strategy()` or `plant::stand_gradient()` as `hyperpar`.

Usage

```
tradeoffs(model, coefficients = list())
```

Arguments

`model`	A plant model name: `"FF16"`, `"TF24"`, `"TF24f"`, `"TF24t"` or `"TF24w"`. The TF24 variants share one set of trade-offs.
`coefficients`	A named list of coefficients that replace the defaults in `tradeoff_coefficients()`. Unknown names are refused.

Details

A trait missing from `m` takes the strategy's value. With `filter = TRUE`, a derived column equal to the strategy's own value in every row is left out, so the strategy keeps its default.

These are the recommended trade-offs. plant keeps its own frozen copy for testing; on any one platform the two give identical numbers.

Value

A function `f(m, s, filter = TRUE)`, carrying `attr(f, "model")`, `attr(f, "coefficients")` and `attr(f, "traits")` (every trait-matrix column it reads). FF16's also carries `attr(f, "input_traits")`, naming the traits it reads that are not parameters of the model.

Examples

```
f <- tradeoffs("TF24", coefficients = list(B_Hv2 = -0.25))
attr(f, "coefficients")$B_Hv2
```

Index

apply_drivers, 2
apply_drivers(), *10*

check_passthrough, 4
check_set *(parameter_set)*, 11

events, *15*

list_sets *(parameter_set)*, 11

met_climatology, 4
met_daily_from_subdaily, 5
met_daily_from_subdaily(), *7*
met_from_monthly, 7
met_from_monthly(), *8*, *9*
met_from_worldclim, 8
met_repeat *(met_climatology)*, 4
met_schema, 10
met_schema(), *2*, *6*, *8*
met_time, 10

parameter_set, 11
pulse_series_fit, 12
pulse_series_fit(), *5*
pulse_series_markov
(pulse_series_fit), 12
pulse_series_poisson, *13*, 14
pulse_series_record
(pulse_series_poisson), 14
pulse_series_regular
(pulse_series_poisson), 14

rainfall_pulse, *13*, *15*
run_set *(parameter_set)*, 11

set_tf24_soil, *16*
soil_widths_graded, 15
solar_day *(solar_declination)*, 17
solar_declination, 17
swdown_clear_sky, 18
swdown_mean_from_noon
(swdown_noon_from_mean), 19
swdown_noon_from_mean, 19
swdown_noon_from_mean(), *6*, *8*, *18*

tradeoff_coefficients, 19
tradeoff_coefficients(), *20*
tradeoff_table
(tradeoff_coefficients), 19
tradeoffs, 20

validate_met *(met_schema)*, 10

worldclim_monthly
(met_from_worldclim), 8