R documentation

of all in 'man/'

May 19, 2016

R topics documented:

`CalculateAll` *Calculate All*

Description

Calculate all scores, given layers and configuration.

Usage

```
CalculateAll(conf, layers, debug = F)
```

Arguments

`conf`	of class `Conf`
`layers`	of class `Layers`
`debug`	print debug messages (default=FALSE)

Details

Performs the following sequence of functions, some of which are [optional]:

functions.R:`Setup()` - execute function `Setup()` if defined in file functions.R. This function typically installs extra packages upon which the other functions in functions.R depend.

1. `CalculatePressuresAll()` - calculate pressures across all goals using pressures_matrix.csv.
2. `CalculateResilienceAll()` - calculate resilience across all goals using resilience_matrix.csv and resilience_weights.csv.
3. goals.csv:`preindex_functions` - execute code in the `preindex_function` column of the goals.csv file based on `order_calculate` using functions defined in functions.R. These functions are usually for calculating the goal's status and trend dimensions, ie the additional dimensions beyond pressures and resilience needed to calculate a goal index score.
4. `CalculateGoalIndex()` - run function for every goal having a status dimension assigned from the `preindex_functions`.
5. goals.csv:`postindex_functions` - execute code in the `postindex_function` column of the goals.csv file based on `order_calculate` using functions defined in functions.R. These functions are usually for goals containing subgoals, ie those without their own directly calculated index scores, but rather scores representing averages of subgoals.
6. regional index - calculate regional index score as weighted mean using goals.csv:`weight`.
7. regional likely future - calculate regional likely future score (ie goal='Index' and dimension='future') across supragoals (ie goals without a `parent` in goals.csv).

functions.R:`PreGlobalScores()` - execute function `PreGlobalScores()` if defined in file functions.R. This function could perform a variety of operations on the regional scores, strategically before calculating the global scores.

8. global (region_id=0) scores - calculate scores for global (region_id=0) with regional values weighted by config.R:`layer_region_areas`.

functions.R:`FinalizeScores()` - execute function `FinalizeScores()` if defined in file functions.R. This function could perform a variety of operations on the regional and global scores.

Value

Returns a data.frame of scores having the following columns:

- *region_id* - unique numeric region identifier, reserving 0 as the region_id for the area-weighted average of the entire study area
- *goal* - the goal code or Index
- *dimension* - the dimension code, one of: status, trend, pressures, resilience, future, score
- *score* - the numeric score: 0-100 for all dimensions, except trend (-1 to 1)

Examples

```
## Not run:
## run a scenario assuming setwd() to directory containing default names for directories and files
## setup
require(ohi)
conf       = Conf('conf')
layers     = Layers(layers.csv = 'layers.csv',
                    layers.dir = 'layers')
## calculate
scores = CalculateAll(conf, layers, debug=T)

## write
write.csv(scores, 'scores.csv', na='', row.names=F)

## End(Not run)
```

CalculateGoalIndex *Calculate Goal Index*

Description

Goal-level computation function to goal score ("component indicators for public goals") based on status, trend, resilience, pressure

Usage

```
CalculateGoalIndex(id, status, trend, resilience, pressure, DISCOUNT = 1,
  BETA = 0.67, default_trend = 0, xlim = c(0, 1))
```

Arguments

id	is the subregion identifier
status	(x) score
trend	(t) score for 5 year outloook
resilience	(r) score

pressure	(p) score
	Constants:
DISCOUNT	is the discount multiplier (i.e., df = 1 - rate)
BETA	is the trend dampening multiplier used in likely future status calculation
default_trend	The default trend value (0) if region has NA.

Details

Parameters:

Value

Returns a data.frame with the input data, a likely future status and OHI score, containing columns: status (x), trend (t), resilience (r), pressure (p), future status (xF) and goal score (score).

Examples

```
## Not run:
## run a model with 50 regions using random data,
## using 5 year 1-percent discount rate and beta=0.67
require(ohi)
d <- ohi.model.goal(id=1:50,
                    status=runif(50, 0, 1),
                    trend=runif(50, -1, 1),
                    resilience=runif(50, 0, 1),
                    pressure=runif(50, 0, 1),
                    DISCOUNT = (1 + 0.01)^-5,
                    BETA = 0.67,
                    default_trend = 0.0)
## view model output
names(d)
d[,c('id','score','xF')]

## End(Not run)
```

CalculatePressures *Calculate the pressures score for each (sub)goal.*

Description

Calculate the pressures score for each (sub)goal.

Usage

```
CalculatePressures(layers, conf, gamma, debug = F)
```

Arguments

layers	object Layers
conf	object Conf
gamma	(optional) if not specified defaults to 0.5

Value

data.frame containing columns 'region_id' and per subgoal pressures score

---

CalculatePressuresAll *Calculate all the pressures score for each (sub)goal.*

Description

Calculate all the pressures score for each (sub)goal.

Usage

```
CalculatePressuresAll(layers, conf, gamma = 0.5, debug = F)
```

Arguments

layers	object Layers
conf	object Conf
gamma	(optional) if not specified defaults to 0.5

Value

data.frame containing columns 'region_id' and per subgoal pressures score

---

CalculatePressuresComponent *Calculate the pressures component of each (sub)goal.*

Description

Calculate the pressures component of each (sub)goal.

Usage

```
CalculatePressuresComponent(eco.pressures, social.pressures,
  c.name = "category", s.name = "region", gamma = 0.5)
```

Arguments

`eco.pressures`	data.frame containing columns 'region', 'category', 'weight', and 'value'
`social.pressures`	data.frame containing columns 'region', and 'value'
`gamma`	(optional) if not specified defaults to 0.5

Value

data.frame containing columns 'region', 'p_E', 'p_S', and 'p_x'

`CalculatePressuresMatrix` *Calculate Pressures Matrix*

Description

The pressures matrix model function computes a pressures weighting matrix based on regional attributes per category.

Usage

```
CalculatePressuresMatrix(alpha, beta, calc = "avg")
```

Arguments

`alpha`	the weighting matrix of the form [category x pressure]. Each rank weight must be an integer between 0 and 3 inclusive, or NA.
`beta`	the aggregation matrix of the form [region_id x category] to collapse across each category.
`calc`	type of calculation, whether avg (default), mean (diff't from avg?) or presence (results in 1 or 0).

Details

Given:

- *g* is the goal or subgoal (e.g., AO, CW, LIV, ECO, ...),
- *i* is the region (e.g., 1, 2, 3, ...),
- *j* is the pressures layer or stressor (e.g., `cc_acid`, `fp_art_lb`, etc.).
- *k* is the category (e.g., habitat, sector, product, etc.)

There may be a component *k* for a given goal *g* such that $p_w(g,i,j,k)$ and $w(g,i,j,k)$.

$$p_w(g,i,j,k) = w(g,i,j,k) * p(i,j)$$

In these cases where there is a component *k* for goal *g*, there's an additional aggregation or formula to calculate $w(g,i,j)$ based on the core rank weight $\alpha(g,j,k)$ from the original pressures matrix (as written in Halpern et al. (2012)) and some region-specific data for each category k $\beta(i,k)$.

This function `CalculatePressuresMatrix` will aggregate a category-specific weighting matrix $\alpha(g,j,k)$ [category x pressure] using region-specific data $\beta(g,i,k)$ into a [region_id x pressure] matrix $w(g,i,j)$ used in `CalculatePressuresScore`, such that:

$$w(g,i,j) = \frac{\sum_k \alpha(g,j,k) * \beta(g,i,k)}{\sum_k \beta(g,i,k)}$$

1. For the CP, CS goals, the weight depends on the extent *A* of habitat *k* in region *i*:

$$\beta(i,k) = A(i,k)$$

2. For the HAB goal, the weight depends on the presence of habitat *k* (i.e., if $A(i,k) > 0$) in region *i*:

$$\beta(i,k) = hasHabitat(i,k)$$

3. For the LIV and ECO goals, the weight depends on the presence of sector *k* if data available for region *i* and sector *k*:

$$\beta(i,k) = hasSector(i,k)$$

4. For the NP goal, the weight depends on the peak dollar value of each product *k* across all years (see w_p from SI Equation S27) if data available for region *i* and product *k*:

$$\beta(i,k) = w_p(i,k)$$

Value

Returns a weight matrix *w* [region_id x pressure] suitable for `CalculatePressuresScore`.

See Also

`CalculatePressuresScore`

`CalculatePressuresScore` *Calculate Pressures Score*

Description

The pressures score is calculated for each region given a weighting matrix for a goal and the individual pressures values.

Usage

```
CalculatePressuresScore(p, w, GAMMA = 0.5, browse = F,
  pressures_categories = list(environmental = c("po", "hd", "fp", "sp", "cc"),
  social = "ss"))
```

Arguments

p	the pressures value matrix [region_id x pressure]. Each score must be a real number between 0 and 1 inclusive, or NA. The pressure names must be of the form *category_pressure* where *category* is one of the categories listed in `ohi.pressure.category`. Use `ss` to denote the social category. `pressure region_id cc_acid cc_sst cc_uv fp_art_hb 1 0.879 0.360 0.764 NA 2 0.579 0.396 0.531 NA 3 0.926 0.235 0.769 NA 4 0.914 0.554 0.795 NA 5 0.860 0.609 0.802 0.001 6 0.871 0.325 0.788 0.001 7 0.846 0.410 0.677 0.000 8 0.806 0.671 0.752 NA 9 0.844 0.595 0.678 NA 10 0.860 0.575 0.781 0.109`
w	the weighting matrix of the form [region_id x pressure]. Each rank weight must be a real number between 0 and 3 inclusive, or NA. `pressure region_id cc_acid cc_sst cc_uv fp_art_hb 1 2 1 0.6 NA 2 2 1 0.5 NA 3 2 1 2.1 NA 4 2 1 3.0 NA 5 2 1 2.8 1 6 2 1 2.2 1 7 2 1 1.3 1 8 2 1 1.7 NA 9 2 1 3.0 NA 10 2 1 1.2 1`
GAMMA	Multiplier used to combine environmental and social pressures.

Details

Each pressure layer $p(i, j)$ is either environmental or social, belongs to a pressures category $K \in \{cc, fp, hd, po, sp, ss\}$, and has a value $(0..1)$ for each region *i* and pressures layer *j*. Each goal has a weight matrix w that has a rank weight between 0 and 3 inclusive, or NoData, for each region *i* and each pressure layer *j* on a per goal *g* basis.

The pressures scores calculations go through 5 steps, using a complex weighting scheme that varies across goals, subgoals, pressures categories, and regions:

- *g* is the goal or subgoal (e.g., AO, CW, LIV, ECO, ...),
- *i* is the region (e.g., 1, 2, 3, ...),
- *j* is the pressures layer or stressor (e.g., `cc_acid`, `fp_art_lb`, etc.).

Calculations

1. Apply weights for each goal *g*, region *i*, and pressure layer *j*: Each weighted pressure $p_w(g, i, j)$ is the pressure layer value $p(i, j)$ per region *i* and pressure layer *j* multiplied by the rank weight $w(g, i, j)$ for that goal *g*, region *i*, and pressure layer *j*. If the $w(g, i, j)$ is NoData or 0, the weighted pressure $p_w(g, i, j)$ is NoData.

$$p_w(g, i, j) = w(g, i, j) * p(i, j)$$

2. Category-level aggregation: The pressures category score p_K is the sum of all p_w within each category, then rescaled to 0..1 using a linear scale range transformation (from 0..3 to 0..1). Any score p_K greater than 1 is capped to 1:

$$p_K(g,i) = \frac{\min(\sum_{j \in K} p_w(g,i,j), 3)}{3}$$

3. Environmental aggregation: The environmental pressures score $p_E(g,i)$ is the weighted sum of p_K(g,i), where each weight is the maximum weight in the pressure category K, and then divided by the sum of the maximum weights:

$$w_{K,max}(g,i) = max(\{\forall_j \in K | w(g,i,j)\})$$

$$p_E(g,i) = \frac{\sum_K w_{K,max}(g,i) p_K(g,i)}{\sum_K w_{K,max}(g,i)}$$

4. Social aggregation: The social pressures score $p_S(g,i)$ is the mean of the *unweighted* social pressure scores $p(i,j)$:

$$p_S(g,i) = \frac{\sum_{j \in S} p(i,j)}{N}$$

5. Gamma combination: The pressures score $p_X(g,i)$:

$$p_X(g,i) = \gamma p_E(g,i) + (1-\gamma) p_S(g,i)$$

Value

Returns a named vector with the pressures score for each named region.

See Also

CalculatePressuresMatrix

Examples

```
## Not run:
  > conf$config$pressures_categories
$environmental
[1] "po" "hd" "fp" "sp" "cc"

$social
[1] "ss"
> p
         pressure
region_id fp_art_hb fp_art_lb fp_com_hb fp_com_lb hd_intertidal
       1      0.122      0.25      0.35     0.395         0.954
       2      0.096      0.94      0.85     0.252         0.649
       3      0.858      0.46      0.84     0.097         0.425
       4      0.814      0.63      0.60     0.672         0.659
```

```
       5      0.247      0.51      0.58     0.941         0.046
       6      0.853      0.34      0.15     0.370         0.385
       7      0.601      0.31      0.39     0.873         0.064
       8      0.355      0.89      0.74     0.159         0.273
       9      0.289      0.94      0.52     0.743         0.094
       10     0.887      0.89      0.87     0.660         0.746
         pressure
region_id hd_subtidal_hb hd_subtidal_sb po_chemicals po_nutrients
       1           0.535          0.651        0.042        0.931
       2           0.454          0.069        0.234        0.025
       3           0.297          0.428        0.970        0.679
       4           0.953          0.485        0.063        0.565
       5           0.963          0.045        0.552        0.828
       6           0.598          0.213        0.907        0.220
       7           0.476          0.641        0.980        0.214
       8           0.285          0.858        0.447        0.793
       9           0.591          0.702        0.719        0.472
       10          0.072          0.431        0.685        0.102
         pressure
region_id sp_alien sp_genetic ss_wgi
       1     0.979      0.761  0.181
       2     0.345      0.091  0.631
       3     0.223      0.986  0.646
       4     0.035      0.078  0.559
       5     0.992      0.643  0.432
       6     0.963      0.416  0.221
       7     0.752      0.627  0.257
       8     0.100      0.245  0.333
       9     0.316      0.373  0.347
       10    0.283      0.224  0.031
> w
         pressure
region_id fp_art_hb fp_art_lb fp_com_hb fp_com_lb hd_intertidal
       1          2         1      0.92         1             1
       2          2         1      0.48         1             1
       3          2         1      2.81         1             1
       4          2         1      1.19         1             1
       5          2         1      2.82         1             1
       6          2         1      1.07         1             1
       7          2         1      1.48         1             1
       8          2         1      0.46         1             1
       9          2         1      0.56         1             1
       10         2         1      0.90         1             1
         pressure
region_id hd_subtidal_hb hd_subtidal_sb po_chemicals po_nutrients
       1               2              2         1.00            1
       2               2              2         0.79            1
       3               2              2         0.37            1
       4               2              2         0.91            1
       5               2              2         1.06            1
       6               2              2         0.72            1
       7               2              2         0.49            1
       8               2              2         1.18            1
```

```
       9              2              2         0.18           1
       10             2              2         0.28           1
         pressure
region_id sp_alien sp_genetic ss_wgi
       1         1          1      1
       2         1          1      1
       3         1          1      1
       4         1          1      1
       5         1          1      1
       6         1          1      1
       7         1          1      1
       8         1          1      1
       9         1          1      1
       10        1          1      1
> p_x <- CalculatePressuresScore(p, w)
> p_x
   1    2    3    4    5    6    7    8    9   10
0.40 0.53 0.68 0.63 0.60 0.43 0.48 0.47 0.50 0.30
> data.frame(region_id=names(p_x), pressure=p_x)
   region_id pressure
1          1     0.40
2          2     0.53
3          3     0.68
4          4     0.63
5          5     0.60
6          6     0.43
7          7     0.48
8          8     0.47
9          9     0.50
10        10     0.30
>
>


## End(Not run)
```

`CalculateResilience` *Calculate the resilience score for each (sub)goal.*

Description

Calculate the resilience score for each (sub)goal.

Usage

```
CalculateResilience(layers, conf, debug = FALSE)
```

Arguments

layers	object Layers
conf	object Conf

Value

data.frame containing columns 'region_id' and per subgoal resilience score

---

CalculateResilienceAll

Calculate all the resilience score for each (sub)goal.

Description

Calculate all the resilience score for each (sub)goal.

Usage

```
CalculateResilienceAll(layers, conf, debug = FALSE)
```

Arguments

layers	object Layers
conf	object Conf

Value

data.frame containing columns 'region_id' and per subgoal resilience score

---

CalculateResilienceComponent

Calculate the Resilience component of each (sub)goal.

Description

Calculate the Resilience component of each (sub)goal.

Usage

```
CalculateResilienceComponent(goal.specific.regulations, ecological.integrity,
  social.integrity, c.name = "category", s.name = "region", gamma = 0.5)
```

Arguments

goal.specific.regulations	(data.frame) contains columns 'region', 'weight', and 'value'
gamma	(numeric) represents the weighting between ecological and social aspects of resilience, defaults to 0.5 (equal weights)

Value

(data.frame)

CalculateResilienceScore

Calculate Resilience Score

Description

The resilience model function computes a resilience score for each region given a weighting matrix for a goal and the individual resilience values.

Usage

```
CalculateResilienceScore(r, t, w = NA, gamma = 0.5,
  resilience_categories = c("environmental", "regulatory", "social"))
```

Arguments

r	the resilience value matrix [region_id x layer]. Each score must be a real number between 0 and 1 inclusive, or NA.
t	the typing vector t[layer] where values are from resilience_categories.
w	the weighting matrix of the form [region_id x layer]. Each rank weight must be a real number >= 0, or NA for even weighting.
gamma	the gamma constant for $r_{i,x}$ calculation.

Details

To calculate Resilience for each goal g and region i ($r(g,i)$) we assess three types of resilience measures *j*: ecological integrity ($Y_E(g,i)$), goal-specific regulations aimed at addressing ecological pressures ($G(g,i)$), and social integrity ($Y_S(g,i)$). The first two measures address ecological resilience while the third addresses social resilience. When all three aspects are relevant to a goal, Resilience is calculated for each goal g and each region *i*:

$$r(g,i) = \gamma * \left(\frac{Y_E(g,i) + G(g,i)}{2} \right) + (1-\gamma) * Y_S(g,i)$$

where each goal *g* is comprised of several resilience layers *j* where w_j is a configuration-time weight to aggregate across resilience categories:

$$G(g,i) = \frac{\sum_{j \in g} w_j G(i,j)}{\sum_{j \in g} w_j}$$

$$Y_E(g,i) = \frac{\sum_{j \in g} Y_E(i,j)}{N}$$

$$Y_S(g,i) = \frac{\sum_{j \in g} Y_S(i,j)}{N}$$

Value

`ohi.model.resilience` returns resilience score for each region. `ohi.model.resilience.matrix` returns a weighting matrix suitable for `ohi.model.resilience`.

Examples

```
## Not run:
> conf$config$resilience_categories
[1] "environmental" "regulatory"    "social"
> b
         layer
region_id fishing-v1 habitat-combo species-diversity-3nm wgi-all
      104       TRUE          TRUE                  TRUE    TRUE
      105       TRUE          TRUE                  TRUE    TRUE
      106       TRUE          TRUE                  TRUE    TRUE
      107       TRUE          TRUE                  TRUE    TRUE
      108       TRUE          TRUE                  TRUE    TRUE
      109       TRUE          TRUE                  TRUE    TRUE
      110       TRUE          TRUE                  TRUE    TRUE
      111       TRUE          TRUE                  TRUE    TRUE
      112       TRUE          TRUE                  TRUE    TRUE
      113       TRUE          TRUE                  TRUE    TRUE
      114       TRUE          TRUE                  TRUE    TRUE
> w
           fishing-v1         habitat-combo species-diversity-3nm
                    2                     2                     1
              wgi-all
                    1
> w < -ohi.model.resilience.matrix(b, w)
> w
         layer
region_id fishing-v1 habitat-combo species-diversity-3nm wgi-all
      104          2             2                     1       1
      105          2             2                     1       1
      106          2             2                     1       1
      107          2             2                     1       1
      108          2             2                     1       1
      109          2             2                     1       1
      110          2             2                     1       1
      111          2             2                     1       1
      112          2             2                     1       1
```

```
      113          2             2                    1       1
      114          2             2                    1       1

> r
         layer
region_id fishing-v1 habitat-combo species-diversity-3nm wgi-all
      104     0.4870        0.4495                0.8679  0.4385
      105     0.5162        0.5905                0.8748  0.2460
      106     0.4811        0.4051                0.8852  0.6465
      107     0.3618        0.2583                0.8260  0.8007
      108     0.5322        0.4703                0.9318  0.5579
      109     0.5053        0.4703                0.9313  0.5579
      110     0.6491        0.5690                0.9239  0.5703
      111     0.3629        0.1562                0.9230  0.6375
      112     0.5670        0.5000                0.9273  0.5718
      113     0.3807        0.2530                0.9339  0.4484
      114     0.6508        0.5690                0.9275  0.5703
> t
           fishing-v1         habitat-combo species-diversity-3nm
         "regulatory"          "regulatory"       "environmental"
              wgi-all
             "social"

> ohi.model.resilience(r, t, w)
   104    105    106    107    108    109    110    111    112    113
0.5533 0.4800 0.6553 0.6844 0.6372 0.6337 0.6684 0.6144 0.6511 0.5369
   114
0.6695

## End(Not run)
```

`CalculateStatusComponent` *Compute a single subgoal.*

Description

Compute a single subgoal.

Usage

```
CalculateStatusComponent(DATA, fun, trend.Years = 5, c.name = "year",
  s.name = "region")
```

Arguments

`DATA`	data.frame containing columns 'region', 'value', and (optionally) 'w'
`fun`	(optional) function for calculating the subgoal value, if not specified it will default to a weighted average

Value

stuff

`CalculateSubgoal` *Compute a single subgoal.*

Description

Compute a single subgoal.

Usage

```
CalculateSubgoal(current.data, eco.pressures, social.pressures, gs.regulations,
  social.integrity, eco.integrity, fun = stats::weighted.mean,
  trend.Years = 5)
```

Arguments

`current.data`	data.frame containing columns 'region', 'value'
`fun`	(optional) function for calculating the subgoal value, if not specified it will default to a weighted average

Value

stuff

`CheckLayers` *Check Layers*

Description

Check all the input layers as defined by layers.csv and update required fields

Usage

```
CheckLayers(layers.csv, layers.dir, flds_id, verbose = T, msg.indent = "  ")
```

Arguments

`layers.csv`	full path to the layers.csv file.
`layers.dir`	full path to the directory containing the layers files.
`flds_id`	character vector of unique identifiers, typically spatial, eg c('region_id', 'country_id', 'saup_id'), described in your `Conf$layers_id_fields`.
`verbose`	if True (default), extra diagnostics are output

Details

The CheckLayers() function iterates through all the layers in layers.csv and updates the following field names, which can be NA for any except flds:

- *fld_id_num* - name of field used as spatial identifier, if numeric
- *fld_id_chr* - name of field used as spatial identifier, if character
- *fld_category* - name of field used as category
- *fld_year* - ame of field used as year
- *fld_val_num* - name of field used as value, from fld_value, if numeric
- *fld_val_chr* - name of field used as value, from fld_value, if character
- *flds* - data fields used for the layer

Additional diagnostic fields are updated:

- *file_exists* - input filename exists
- *year_min* - minimum year, if year present
- *year_max* - maximum year, if year present
- *val_min* - minimum value, if numeric
- *val_max* - maximum value, if numeric
- *val_0to1* - TRUE if value ranges between 0 and 1
- *flds_unused* - unused fields from input file when guessing prescribed field names (aboves)
- *flds_missing* - fields expected, as given by Layers units, and not found
- *rows_duplicated* - given the combination of all row-identifying fields (and excluding value fields), the number of rows which are duplicates
- *num_ids_unique* - number of unique ids, as provided by just the unique instances of the fld_id

Value

warning messages

Examples

```
## Not run:
  CheckLayers(layers.csv, layers.dir, c('rgn_id','cntry_key','saup_id'))

## End(Not run)
```

compare_scores_df *Compares scores*

Description

Combine two scores.csv files and calculate difference.

Usage

```
compare_scores_df(a_csv, b_csv, r_csv, g_csv)
```

Arguments

a_csv	scores.csv for A
b_csv	scores.csv for B
r_csv	region labels, ie layers/rgn_labels.csv
g_csv	goals, ie conf/goals.csv

Details

Returns a data frame with calculated differences sorted by global (region_id=0), Index, score, goal, dimension, absolute score, is.na(a), is.na(b).

Conf *Conf reference class.*

Description

Conf reference class.

Usage

```
Conf(...)
```

Arguments

dir	path to directory containing necessary files

Details

To create this object, `Conf(dir)`. The `dir` is expected to have the following files:

- *config.R*
- *functions.R*
- *goals.csv*
- *pressures_matrix.csv*
- *resilience_matrix.csv*
- *resilienceweights.csv*

See also `Conf_write()` to write the configuration back to disk.

Value

object reference class of Config containing:

- *config*
- *functions*
- *goals*
- *pressures_matrix*
- *resilience_matrix*
- *resilienceweights*

`Conf-class` *Conf reference class.*

Description

Conf reference class.

Arguments

`dir`	path to directory containing necessary files

Details

To create this object, `Conf(dir)`. The `dir` is expected to have the following files:

- *config.R*
- *functions.R*
- *goals.csv*
- *pressures_matrix.csv*
- *resilience_matrix.csv*
- *resilienceweights.csv*

See also `Conf_write()` to write the configuration back to disk.

Value

object reference class of Config containing:

- *config*
- *functions*
- *goals*
- *pressures_matrix*
- *resilience_matrix*
- *resilienceweights*

Conf_write *Write the Conf to disk*

Arguments

dir	path to directory where the Conf files should be output

Details

Use this function to write the configuration to disk, like so `conf$write(dir)`. This is useful for modifying and then reloading with `Conf(dir)`.

---

gapfill_georegions *Gapfill using georegional means*

Description

Gapfill using georegional means, providing the finest possible resolution from 3 hierarchies (r2 > r1 > r0) derived from United Nations geoscheme.

Usage

```
gapfill_georegions(data, georegions, fld_id = intersect(names(data),
  names(georegions)), fld_year = ifelse("year" %in% names(data), "year",
  NA), fld_value = setdiff(names(data), c(fld_id, fld_weight, "year")),
  georegion_labels = NULL, fld_weight = NULL, rgn_weights = NULL,
  ratio_weights = FALSE, gapfill_scoring_weights = c(r0 = 1, r1 = 0.8, r2 =
  0.5, v = 0), r0_to_NA = TRUE, attributes_csv = NULL)
```

Arguments

data	data.frame to gapfill having at least fields: fld_id and fld_value, and optionally fld_weight
georegions	data.frame having at least fields: fld_id and r0, r1, and r2 with georegion id values
fld_id	common spatial id field (eg region_id or country_key) between data and georegions
fld_year	optional year field in data
fld_value	value to gapfill in data
georegion_labels	with same dimensions as georegions having fields: r0_label, r1_label, r2_label and v_label
fld_weight	optional weighting field in data
rgn_weights	data frame of weights, expecting rgn_id in first column and weight in second
ratio_weights	if TRUE, multiply the gapfilled value by the ratio of the region's weight to the regional average weight. Defaults to FALSE. IMPORTANT to set to TRUE if dealing with values that SUM!
gapfill_scoring_weights	used to determine gapfilling scoreset. should range 0 to 1. defaults to c('r0'=1, 'r1'=0.8, 'r2'=0.5,
r0_to_NA	assign value of NA if only georegional average availabe at the global level (r0). defaults to True.
attributes_csv	optional path and filename to save attribute table. defaults to NULL

Details

Gapfill using georegional means, providing the finest possible resolution from 3 hierarchies (r2 > r1 > r0).

The gapfill score (z_g_score) in the attribute table is formulated such that the higher the score, the more gapfilling performed. The maximal gapfill score is based on gapfilling at the global level (r0=1) and least if no gapfilling performed (ie z = v). But then some regional averages are applied with only a few regional values while others might have all but the gapfilled region available. To account for this aspect, the difference between the next finer level's weight is multiplied by the percent regions and subtracted from the level's weight, like so:

```
gapfill_scoring_weights[z_level] - z_n_pct * diff(gapfill_scoring_weights[z_level, z_level_finer])
```

Value

Returns a data.frame of having all the fld_id from georegions filled in the following columns:

- fld_id - spatial id (eg region_id or country_key).
- fld_value - the gapfilled value (eg score).

The returned data.frame also has an attribute "gapfill_georegions" which shows the calculated georegional means and which levels were chosen:

- r0 - georegional id for level 0, ie global.

- r1 - georegional id for level 1.
- r2 - georegional id for level 2, the finest resolution of georegions.
- id - spatial id (eg region_id or country_key).
- w - weight used to apply weighted.mean. Defaults to 1 if not supplied as fld_weight parameter.
- v - original fld_value in data
- r2_v - weighted.mean for level 2
- r1_v - weighted.mean for level 1
- r0_v - weighted.mean for level 0 (global)
- r2_n - count of regions available for level 2
- r1_n - count of regions available for level 1
- r0_n - count of regions available for level 0
- r2_n_notna - count of region values that are not NA for level 2
- r1_n_notna - count of region values that are not NA for level 1
- r0_n_notna - count of region values that are not NA for level 0
- z_level - finest level available
- z_ids - ids for regions that are not NA which contributed to the score
- z_n - count of input values for finest level available
- z_n_pct - percent of region values that are not NA over all possible [0 to 1]
- z_g_score - gapfilling score (see details)
- z - weighted.mean for finest level available

Examples

```
## Not run:
## setup
require(ohicore)

# gapfill
g = gapfill_georegions(data, georegions, fld_weight='w_sum')

# show result and table
head(g)
head(attr(g, 'gapfill_georegions'))

## End(Not run)
```

`Layers` *Layers reference class.*

Description

Layers reference class.

Usage

```
Layers(...)
```

Arguments

`layers.csv`	path to comma-seperated value file with row of metadata per layer
`layers.dir`	path of directory containing individual layer files

Details

To instantiate this object, `Layers(layers.csv, layers.dir)` is used. The `layers.csv` is expected to have the following columns:

- *layer* - unique layer identifier (no spaces or special characters)
- *targets* - a space delimited list of targets (goal code, 'Pressures', 'Resilience' or 'Regions') for which this layer is applied
- *name* - name of the variable
- *description* - detailed description
- *units* - units of the value
- *citation* - reference for documentation, typically a heading code for a supplemental document
- *filename* - the csv data file for the layer
- *fld_value* - required field in the layer csv file containing the value, which is often best named as a shorthand for the units without spaces or special characters

The layers.dir directory should contain all the csv filenames listed in the layers.csv file.

Value

object (non-instantiated) reference class of Layers containing

- *meta* - metadata data frame of original layers.csv
- *data* - named list of data frames, one per layer
- *targets* - named list of character vector indicating a layer's targets, goal (status, trend) or dimension (pressures, resilience)

`Layers-class` *Layers reference class.*

Description

Layers reference class.

Arguments

`layers.csv`	path to comma-seperated value file with row of metadata per layer
`layers.dir`	path of directory containing individual layer files

Details

To instantiate this object, `Layers(layers.csv, layers.dir)` is used. The `layers.csv` is expected to have the following columns:

- *layer* - unique layer identifier (no spaces or special characters)
- *targets* - a space delimited list of targets (goal code, 'Pressures', 'Resilience' or 'Regions') for which this layer is applied
- *name* - name of the variable
- *description* - detailed description
- *units* - units of the value
- *citation* - reference for documentation, typically a heading code for a supplemental document
- *filename* - the csv data file for the layer
- *fld_value* - required field in the layer csv file containing the value, which is often best named as a shorthand for the units without spaces or special characters

The layers.dir directory should contain all the csv filenames listed in the layers.csv file.

Value

object (non-instantiated) reference class of Layers containing

- *meta* - metadata data frame of original layers.csv
- *data* - named list of data frames, one per layer
- *targets* - named list of character vector indicating a layer's targets, goal (status, trend) or dimension (pressures, resilience)

mapvalues *mapvalues*

Description

Replace specified values with new values, in a vector or factor. This is copied from plyr.

Usage

```
mapvalues(x, from, to, warn_missing = TRUE)
```

Arguments

x	the factor or vector to modify
from	a vector of the items to replace
to	a vector of replacement values
warn_missing	print a message if any of the old values are not actually present in x

Details

#' If x is a factor, the matching levels of the factor will be replaced with the new values.

The related revalue function works only on character vectors and factors, but this function works on vectors of any type and factors.

Value

Returns a vector with new values.

name_to_rgn *Get scenarios*

Description

Get scenarios from Github.

Usage

```
name_to_rgn(d, fld_name = "country", flds_unique = fld_name,
  fld_value = "value", collapse_fxn = c("sum_na", "mean",
  "weighted.mean")[1], collapse_csv = NULL, collapse_flds_join = NULL,
  dir_lookup = "~/github/ohiprep/src/LookupTables",
  rgn_master.csv = file.path(dir_lookup, "eez_rgn_2013master.csv"),
  rgn_synonyms.csv = file.path(dir_lookup, "rgn_eez_v2013a_synonyms.csv"),
  add_rgn_name = F, add_rgn_type = F)
```

Arguments

d	dataset
fld_name	field name of the region from the dataset
flds_unique	field name for the dataset
fld_value	field with value, defaults to 'value'
collapse_fxn	function to collapse duplicate regions into one (example: China, Macau, Hong Kong)
collapse_csv	optional .csv file provided to collapse duplicate regions
collapse_flds_join	optional list of fields identified to collapse duplicate regions
dir_lookup	directory of name-to-region look up tables
rgn_master.csv	.csv file of eez-to-region combinations
rgn_synonyms.csv	.csv file of synonyms of eez-to-region combinations
add_rgn_name	T or F whether to include a column with the region name
add_rgn_type	T of F whether to include the region type (eez...)

Details

This function translates name to region id with a lookup.

PlotFlower *Plot flower plot*

Description

Plot flower plot

Usage

```
PlotFlower(lengths, widths, labels, disk = 0.5, max.length, center = NULL,
  main = NULL, fill.col = NULL, plot.outline = TRUE,
  label.offset = 0.15, xlim = c(-1.2, 1.2), ylim = c(-1.2, 1.2),
  uin = NULL, tol = 0.04, cex = 1, bty = "n", lty = 1,
  label.col = "black", label.font = 3, label.cex = NULL, ...)
```

Arguments

lengths	length of petal outward to extent of circle
widths	width of petal
labels	petal label outside of circel
disk	relative radius of a central donut hole
max.length	...

center	center value
main	middle value
fill.col	fill colors
plot.outline	size of plot outline
label.offset	label offset
xlim	formatting
ylim	formatting
uin	formatting
tol	formatting
cex	size of middle text
bty	formatting
lty	line thickness
label.col	label color
label.font	label font
label.cex	size of label text

Value

Generate something akin to a rose plot in which the width and length of each petal are directly specified by the user. Or to put it differently, this is somewhat like a pie chart in which the radius of each wedge is allowed to vary (along with the angular width, as pie charts do). As an additional enhancement, one can specify a central disk of arbitrary radius (from 0 to 1, assuming that the plot itself is scaled to the unit circle), in which case the petal heights are always measured from the edge of the disk rather than the center of the circle; if desired, text can be added in the center.

Although this kind of plot may already be well known in some circles (no pun intended), I haven't seen it clearly defined or labeled anywhere, so I'm anointing it an 'aster' plot because its component parts are reminiscent of composite flower morphology.

The 'lengths' dictates how far out each petal extends, 'widths' dictates the (angular) width of each petal, and 'disk' gives the relative radius of a central donut hole. If no widths are provided, all petals will have equal widths. Additional function arguments can also control whether petals are labeled, whether the petal lengths are rescaled to the maximum score or to a user-input score, whether spokes delineating each petal are extended to an outer circle, and more. I also wrote a quick convenience wrapper for creating a legend plot.

Note that the function here is a repurposed and very heavily modified version of the windrose() function contained in the 'circular' package, although sufficiently rewritten so as not to depend on any functionality in that package.

Author(s)

Created by Jim Regetz. Slight modifications by Darren Hardy and Ben Best.

Examples

```
## Not run:
# generate some fake data
set.seed(1)
scores <- sample(1:10)
weights <- sample(1:10)
labels <- paste(LETTERS[1:10], "X", sep="")

# do some plots
par(mfrow=c(2,2), xpd=NA)
aster(lengths=scores, widths=weights, disk=0, main="Example 1",
    plot.outline=FALSE)
aster(lengths=scores, widths=weights, labels=labels, main="Example 2",
    lty=2, fill.col="gray", plot.outline=FALSE)
aster.legend(labels=labels, widths=weights)
aster(lengths=scores, widths=weights, disk=0.5, main="Example 3",
    center="Hello world")

## End(Not run)
```

`read_git_csv` *Read CSV from local Git repository*

Description

Read CSV from local Git repository.

Usage

```
read_git_csv(repo, hex, path, ...)
```

Arguments

`repo`	path to local repository
`hex`	SHA hex of commit
`path`	to csv file with the repository as root

Details

If you have trouble running this function, please make sure: 1) your path resolves to a local Git repository, 2) the hex is valid (see history in RStudio Git or on Github) and 3) you have the latest git2r (try `devtools::install_github('ropensci/git2r')`).

Examples

```
## Not run:
# get csv from github repository by SHA hex of commit
d = read_git_csv('~/github/ohi-global', 'a81a8213', 'scores.csv')
head(d)

## End(Not run)
```

ScoreScaling *Score Scaling Functions*

Description

Scoring functions

Usage

```
score.rescale(x, xlim = NULL, method = "linear", ...)
```

Arguments

x	A numeric vector of data.
xlim	The scoring range. If null, derives range from data.
method	Only 'linear' is supported.
...	Arguments for min, max, pmin, pmax.
p	A percentage buffer to add to the maximum value.

Value

Returns scores.

See Also

min, max, pmin, pmax

Examples

```
score.max(c(0.5, 1, 2))
score.max(c(0.5, 1, 2), p=0.25)
score.rescale(c(0.5, 1, 2))
score.clamp(c(-0.5, 1, 2))
score.clamp(c(-0.5, 1, 2), xlim=c(-1, 1))
```

`SelectLayersData` *Select Layers to Data*

Description

Select Layers to Data

Usage

```
SelectLayersData(object, targets = NULL, layers = NULL, cast = TRUE,
  narrow = FALSE, expand.time.invariant = FALSE)
```

Arguments

`object`	instance of Layers class
`targets`	specifies the targets of layers to be selected, defaulting to `c('regions')`
`layers`	specifies the layers to be selected. If given as a named character vector, then layers get renamed with new names as values, and old names as names per `plyr::rename`
`narrow`	narrow the resulting data frame to just the fields containing data (as described by *flds* in the default wide result) #@param expand.time.invariant for layers without a year column, populate the same value throughout all years where available in other layer(s) #@param cast whether to cast the resulting dataset, or leave it melted, defaults to TRUE

Details

If neither targets or layers are specified then all layers are returned. If targets and layers are specified, then the union of the two sets of layers are returned, with any renamed layers renamed.

Value

data.frame with the merged data of selected layers having the following fields:

- *layer* - layer name, possibly renamed
- *layer0* - original layer name, if fed a named character vector to layers
- *id_num* - numeric id
- *id_chr* - character id
- *id_name* - fieldname of id in original layer csv file
- *category* - category
- *category_name* - fieldname of character in original layer csv file
- *year* - year
- *val_num* - numeric value
- *val_chr* - character value
- *val_name* - fieldname of value in original layer csv file
- *flds* - data fields used for the layer

shp_to_geojson *Create GeoJSON from Shapefile*

Description

Create GeoJSON file needed for interactive map in Shiny app

Usage

```
shp_to_geojson(shp, js, geojson = sprintf("%s.geojson",
  tools::file_path_sans_ext(js)))
```

Arguments

shp	path to shapefile with .shp extension, needs rgn_id and rgn_name fields
js	path to output javascript file with variable 'regions' of geojson content
geojson	path to output GeoJSON file. defaults to *.geojson of *.js file.

Details

Uses rgdal to write GeoJSON.

SpatialSchemes *SpatialSchemes reference class.*

Description

SpatialSchemes reference class.

Usage

```
SpatialSchemes(...)
```

Value

object (non-instantiated) reference class of SpatialSchemes

`trace_git_csv_value` *Trace Value from CSV through history of local Git repository*

Description

Trace Value from CSV through history of local Git repository

Usage

```
trace_git_csv_value(repo, csv, subset_str, select, verbose = T)
```

Arguments

`repo`	path to repository on local filesystem
`csv`	path to csv file with the repository as root
`subset_str`	subset argument to the function `subset` quoted as string to extract row of data from csv
`select`	field to select from subsetted row

Details

If you have trouble running this function, please make sure: 1) your path resolves to a local Git repository, 2) you have the latest git2r (try `devtools::install_github('ropensci/git2r')`).

Value

data.frame having columns: hex, when, message, v.

Examples

```
## Not run:
# trace the value for a csv from github repository
d = trace_git_csv_value('~/github/ohicore', 'inst/extdata/scores.Global2013.www2013.csv', "goal=='ECO' & dimens
head(d)

## End(Not run)
```

Index