# Package: kwb.rabimo (via r-universe)

November 20, 2024

**Title** R Implementation of Water Balance Model Abimo

**Version** 1.0.1

**Description** The code in this package has been transferred from the C++ code of ABIMO 3.3: Water Balance Model for Urban Areas (https://github.com/KWB-R/abimo/).

**License** MIT + file LICENSE

**URL** https://github.com/KWB-R/kwb.rabimo

**BugReports** https://github.com/KWB-R/kwb.rabimo/issues

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Suggests** covr

**Imports** dplyr, kwb.abimo, kwb.utils, magrittr, methods, parallel, xml2

**Remotes** github::kwb-r/kwb.abimo, github::kwb-r/kwb.utils

**Config/pak/sysreqs** git libarchive-dev libxml2-dev libssl-dev

**Repository** https://kwb-r.r-universe.dev

**RemoteUrl** https://github.com/KWB-R/kwb.rabimo

**RemoteRef** HEAD

**RemoteSha** 911e0ca250ded116061d27719ef83056e579ed99

## Contents

---

`abimo_config_to_config` *Convert Abimo Configuration to List*

---

## Description

Convert Abimo Configuration to List

## Usage

```
abimo_config_to_config(abimo_config)
```

## Arguments

| | |
|---|---|
| `abimo_config` | as returned by `kwb.abimo:::read_config` |

## Value

list with elements `"potential_evaporation"`, `"runoff_factors"`, `"bagrov_values"`, `"diverse"`, `"result_digits"`

---

`actual_evaporation_waterbody_or_pervious` *Calculate Actual Evapotranspiration for Waterbodies or Pervious Areas*

---

## Description

Calculate Actual Evapotranspiration for Waterbodies or Pervious Areas

## Usage

```
actual_evaporation_waterbody_or_pervious(
  usage_tuple,
  potential_evaporation,
  soil_properties,
  precipitation,
  dbg = TRUE,
  ...,
  digits = NULL
)
```

## Arguments

| | |
|---|---|
| usage_tuple | list as returned by get_usage_tuple |
| potential_evaporation | potential evaporation in mm |
| soil_properties | list as returned by get_soil_properties |
| precipitation | precipitation in mm |
| dbg | logical indicating whether or not to show debug messages |
| ... | further arguments passed to real_evapo_transpiration such as run_parallel, blocksize |
| digits | optional. If given, the BAGROV parameter values are rounded to this number of digits. This reduces the number of BAGROV curves that need to be calculated and thus improves the performance (by reducing the precision of the output) |

---

# BERLIN_TYPES_TO_USAGE_YIELD_IRRIGATION

*Assignment between (NUTZUNG, TYP) and (usage, yield, irrigation)*

---

## Description

The following tables are read from csv files and then merged:

## Usage

```
BERLIN_TYPES_TO_USAGE_YIELD_IRRIGATION
```

## Format

An object of class data.frame with 180 rows and 5 columns.

## Details

berlin_tuples.csv table of different occurring (usage, yield, irrigation) tuples

berlin_type_tuple_groups.csv assignments between berlin_type (input column "TYP") and tuples

berlin_usage_to_type_tuple_groups.csv assignments between berlin_usage (input column "NUTZUNG") and assignments between berlin_type (input column "TYP") and tuple

---

`call_with_data` *Call a Function with Argument Combinations from a Data Frame*

---

## Description

Call a Function with Argument Combinations from a Data Frame

## Usage

```
call_with_data(
  FUN,
  data,
  ...,
  threshold = 0.5,
  SIMPLIFY = TRUE,
  USE.NAMES = TRUE
)
```

## Arguments

| | |
|---|---|
| `FUN` | function to be called |
| `data` | data frame with one column per argument of `FUN` |
| `...` | further (constant) arguments to `FUN` that are passed to `mapply` via `MoreArgs` |
| `threshold` | if the ratio of unique value combinations in the relevant columns in data to all value combinations in these columns is below this threshold value then FUN will be called only with the unique value combinations. This should increase performance. |
| `SIMPLIFY` | passed to`mapply`, default: `TRUE` |
| `USE.NAMES` | passed to`mapply`, default: `TRUE` |

## Value

vector of length `nrow(data)` with the result values returned by `FUN`

## Examples

```
combis <- expand.grid(x = 1:2, y = c(10, 20, 30))
combis

call_with_data(`+`, combis)
```

`get_potential_evaporation` *Provide Data on Potential Evaporation*

## Description

Provide Data on Potential Evaporation

## Usage

```
get_potential_evaporation(is_waterbody, district, lookup)
```

## Arguments

| | |
|---|---|
| `is_waterbody` | (vector of) logical indicating whether a block area is of type (from the type/yield/irrigation tuple) "waterbody" |
| `district` | (vector of) integer indicating the district number of the plot area (from the original input column "BEZIRK") |
| `lookup` | data frame with key columns `is_waterbody`, `district` and value columns `etp`, `etps`. A data frame of the required structure is returned by `abimo_config_to_config` in list element `"potential_evaporation"` |

## Examples

```
## Not run:
config <- abimo_config_to_config(kwb.abimo:::read_config())
get_potential_evaporation(
  is_waterbody = TRUE,
  district = 1,
  lookup = config$potential_evaporation
)

## End(Not run)
```

`get_precipitation` *Provide Information on Precipitation*

## Description

Provide Information on Precipitation

## Usage

```
get_precipitation(precipitation_year, precipitation_summer, correction_factor)
```

### Arguments

`precipitation_year`
: precipitation per year in mm

`precipitation_summer`
: precipitation within summer period in mm

`correction_factor`
: correction factor

### Value

list with elements `per_year`, `in_summer`

### Examples

```
get_precipitation(600, 300, 0.8)
```

---

## `get_soil_properties` *Calculate Soil Properties*

---

### Description

Provide variables that are relevant to calculate the actual evaporation for unsealed areas

### Usage

```
get_soil_properties(
  usage,
  yield,
  depth_to_water_table,
  field_capacity_30,
  field_capacity_150,
  default_for_waterbodies = NA,
  dbg = FALSE
)
```

### Arguments

`usage`
: usage string, one of "vegetationless_D", "waterbody_G", "horticultural_K", "agricultural_L", "forested_W"

`yield`
: yield class

`depth_to_water_table`
: depth to water table

`field_capacity_30`
: field capacity in 30 cm depth

`field_capacity_150`
: field capacity in 150 cm depth

| | |
|---|---|
| default_for_waterbodies | value to be used for waterbodies. Default: NA |
| dbg | logical indicating whether or not to show debug messages |

---

`get_usage_tuple` *Get Usage Tuple (Usage, Yield, Irrigation) from NUTZUNG and TYP*

---

## Description

Get Usage Tuple (Usage, Yield, Irrigation) from NUTZUNG and TYP

## Usage

```
get_usage_tuple(usage, type, include_inputs = FALSE)
```

## Arguments

| | |
|---|---|
| usage | value of column NUTZUNG in input data frame |
| type | value of column TYP in input data frame |
| include_inputs | logical indicating whether or not to include the input values in the output |

## Value

list with elements `usage`, `yield`, `irrigation`

## Examples

```
get_usage_tuple(10, 10)
get_usage_tuple(10, 1:3)
```

---

`index_string_to_integers` *Convert String of Integer Ranges to Vector of Integer*

---

## Description

Convert e.g. "1,4-6,10-11,20" to c(1L, 4L, 5L, 6L, 10L, 11L, 20L)

## Usage

```
index_string_to_integers(x, splits = c(",", "-"))
```

### Arguments

| | |
|---|---|
| x | vector of character of length one representing a string of integer ranges, e.g. "1,4-6,10-11,20" |
| splits | characters at which to 1. split x into range strings, 2. split the range strings into begin and end values of the ranges. Default: c(",", "-") |

### Value

vector of integer

### Examples

```
index_string_to_integers("1,4-6,10-11,20")
```

---

## list_to_data_frame_with_keys

*Convert List of Similar Flat Sublists to a Data Frame*

---

### Description

Convert List of Similar Flat Sublists to a Data Frame

### Usage

```
list_to_data_frame_with_keys(x, key_name, key_pattern, convert = identity)
```

### Arguments

| | |
|---|---|
| x | list of similar flat lists, i.e. lists that have list elements with the same names and list elements that all have length one |
| key_name | name of column in the returned data frame that will contain the integer values that are constructed from the element names in x |
| key_pattern | regular expression matching all element names in x. The expression must contain one pair of parentheses enclosing the part that is to be used as key, e.g. "element_([0-9]+)" |
| convert | function to be applied to the (character) key. Set e.g. convert = as.integer to generate integer keys. Default: identity |

### Value

data frame with keys in a column named according to key_name and value columns according to the list elements in the sublists of x

## Examples

```
list_to_data_frame_with_keys(
  x = list(
    element_1 = list(a = 100, b = 10),
    element_2 = list(a = 200, b = 20)
  ),
  key_name = "element",
  key_pattern = "element_([0-9]+)",
  convert = as.integer
)
```

---

`prepare_input_data` *Prepare Input Data: Rename, Add Columns*

---

## Description

Rename columns from ABIMO 3.2 original names to ABIMO 3.3 internal names

## Usage

```
prepare_input_data(input_data)
```

## Arguments

| | |
|---|---|
| `input_data` | data frame with columns REGENJA, REGENSO, NUTZUNG, TYP, BEZIRK, FLGES, STR_FLGES, PROBAU, PROVGU, VGSTRASSE, KAN_BEB, BELAG1, BELAG2, BELAG3, BELAG4, KAN_VGU, STR_BELAG1, STR_BELAG2, STR_BELAG3, STR_BELAG4, KAN_STR, FLUR, FELD_30, FELD_150 |

## Value

`input_data` with columns renamed and additional columns (e.g. ratios calculated from percentages, (main) usage, yield, irrigation)

---

`range_to_seq` *Sequence of Values Between the Range of Values in a Given Vector*

---

## Description

Sequence of Values Between the Range of Values in a Given Vector

## Usage

```
range_to_seq(x, by = 1)
```

### Arguments

| | |
|---|---|
| x | vector of values from which to take the range |
| by | increment of seqence |

### Value

sequence of values between `min(x)` and `max(x)` with increment `by`

---

`real_evapo_transpiration` *Calculate Actual Evapotranspiration with Bagrov*

---

### Description

Calculate Actual Evapotranspiration with Bagrov

### Usage

```
real_evapo_transpiration(
  precipitation,
  potential_evaporation,
  bagrov_parameter,
  x_ratio = NULL,
  FUN_y_ratio = y_ratio_3,
  ...
)
```

### Arguments

| | |
|---|---|
| `precipitation` | precipitation in mm |
| `potential_evaporation` | potential evaporation in mm |
| `bagrov_parameter` | Bagrov parameter (n-value) |
| `x_ratio` | optional. Instead of `precipitation` and `potential_evaporation` the quotient of both may be passed to this function. The idea is to calculate the quotient out of the function and to reuse the quotient instead of recalculating it. |
| `FUN_y_ratio` | function to be called to calculate the y_ratio(s) from the given x_ratio(s). Default: `kwb.rabimo:::y_ratio_3` |
| `...` | further arguments passed to `FUN_y_ratio` |

### Value

estimated actual evapotranspiration in mm

`run_rabimo` *Run R-Abimo, the R-implementation of Water Balance Model Abimo*

## Description

Run R-Abimo, the R-implementation of Water Balance Model Abimo

## Usage

```
run_rabimo(input_data, config, simulate_abimo = TRUE)
```

## Arguments

| | |
|---|---|
| `input_data` | data frame with columns as required by Abimo |
| `config` | configuration object (list) as returned by `kwb.abimo:::read_config()` |
| `simulate_abimo` | logical of length one indicating whether or not to simulate exactly what Abimo does (including obvious errors!). Default: `TRUE`! |

## Value

data frame with columns as returned by Abimo

---

`yearly_height_to_volume_flow`
*Convert Yearly Height (mm) to Volume Flow (unit?)*

## Description

Convert Yearly Height (mm) to Volume Flow (unit?)

## Usage

```
yearly_height_to_volume_flow(height, area)
```

## Arguments

| | |
|---|---|
| `height` | height in mm |
| `area` | area in square metres |

`y_ratio_3` *Lookup y_ratio for given x_ratio on a BAGROV curve*

## Description

Lookup y_ratio for given x_ratio on a BAGROV curve

## Usage

```
y_ratio_3(
  bagrov_parameter,
  x_ratio,
  min_size_for_parallel = 10L,
  use_abimo_algorithm = FALSE
)
```

## Arguments

| | |
|---|---|
| `bagrov_parameter` | (vector of) BAGROV parameter(s) |
| `x_ratio` | (vector of) x-ratio(s) (between precipitation and potential evaporation) for which to look up the corresponding y-ratio(s) (between actual evaporation and potential evaporation) on the BAGROV curve(s) |
| `min_size_for_parallel` | minimum number of BAGROV curves to start parallel processing |
| `use_abimo_algorithm` | whether or not to use the original algorithm that is implemented in the C++ code (converted to R: `kwb.rabimo:::yratio_cpp`). Default: `FALSE` |

# Index