extended R documentation

of function `sorcering()`

from package `'sorcering'`

December 17, 2023

Version	1.0.0.1
Title	SORCERING: Soil ORganic Carbon & CN Ratio drIven Nitrogen modellinG framework
Authors	Marc Scherstjanoi* & René Dechow◇ * Thünen Institute of Forest Ecosystems, Eberswalde, Germany ◇ Thünen Institute of Climate-Smart Agriculture, Braunschweig, Germany
Maintainer	Marc Scherstjanoi <marc.scherstjanoi@thuenen.de>
LazyData	true
Depends	R (>= 3.5.0)
License	GLP (>= 2)
Imports	Rcpp (>= 1.0.6)
LinkingTo	Rcpp, RcppArmadillo
NeedsCompilation	yes

Description

`SORCERING` can be used to model the fate of soil organic carbon (SOC) and soil organic nitrogen (SON) and to calculate N mineralisation rates. It provides a framework that numerically solves differential equations of SOC models based on first-order kinetics. An SOC model can be simply defined or a predefined existing SOC model can be chosen and then run to predict the temporal development of SOC. Beyond this, `SORCERING` determines the fluxes of SON and N mineralisation / immobilisation. Basic inputs are (1) the model parameters of a given SOC model expressed as the C transfer matrix (including information on decomposition and transfer rates between model pools), (2) either the initial distributions of C and N among model pools as a direct input or time series of at least three C and N measurement points with which these initial distributions can be calculated using linear regression, and (3) time series of C and N inputs and rate modifying environmental factors. In case a predefined SOC model is used, instead of model parameters and time series of rate modifying factors, model-specific environmental and stand data must be passed for the calculation of decomposition and transfer rates. Moreover, `SORCERING` provides the option to cover statistical uncertainties by using stochastic repetitions for many input data and parameters. The fourth-order Runge-Kutta algorithm is used to numerically solve the system of differential equations.

Contents

Usage

```
sorcering(       A = NULL,
                 tsteps = "monthly",
                 t_sim = 2,
                 C0 = NULL,
                 N0 = NULL,
                 Cin = NULL,
                 Nin = NULL,
                 Cin_wood = NULL,
                 Nin_wood = NULL,
                 wood_diam = NULL,
                 xi = NULL,
                 env_in = NULL,
                 site = NULL,
                 theta = NULL,
                 theta_unc = NULL,
                 theta_n_unc = 1,
                 meas_data = NULL,
                 t_sim_sl = 2,
                 A_sl = NULL,
                 C0_sl = NULL,
                 N0_sl = NULL,
                 Cin_sl = NULL,
                 Nin_sl = NULL,
                 Cin_wood_sl = NULL,
                 Nin_wood_sl = NULL,
                 wood_diam_sl = NULL,
                 xi_sl = NULL,
                 env_in_sl = NULL,
                 site_sl = NULL,
                 sitelist = NULL,
                 meas_data_sl = NULL,
                 calcN = FALSE,
                 calcNbalance = FALSE,
                 calcN0 = FALSE,
                 calcC0 = FALSE,
                 calcCN_fast_init = FALSE,
                 CTool_input_raw = FALSE,
                 RothC_Cin4C0 = FALSE,
                 C0_fracts = NULL,
                 multisite = FALSE,
                 pooltypes = NULL,
                 CN_fast_init = 40,
                 CN_bio = 9,
                 CN_fast_init_sl = NULL,
                 CN_bio_sl = NULL,
                 init_info = FALSE,
                 model = "")
```

Arguments

In addition to the arguments descriptions below, Fig. 1 provides a visualisation of the list structure and Fig. 2 provides a visualisation of the content structure of the arguments.

`A`	square matrix. Transfer matrix typical for SOC modelling. Defines number of pools, decomposition and transfer rates. n $\times$ n elements with n = number of pools. Diagonal values are decomposition rates [yr^{-1}]. Off-diagonals represent the transfer between pools [yr^{-1}]. Only used when `model is NULL`.
`tsteps`	character string indicating the type of simulation time steps. Valid options are `"annually"`, `"monthly"` (recommended) or `"weekly"`. Ensures that the rate modifying factors (passed through `xi` or used by a predefined model) are adjusted by dividing them by 1, 12, and 52 respectively. Furthermore ensures, that environment-specific information (passed through `env_in`) takes into account the time reference of the modelling.
`t_sim`	integer number of simulation time steps. Must correspond to the number of rows of `Cin`, `Nin` and `xi`.
`C0`	either vector with a length equal to the number of pools or scalar. If vector, initial soil organic carbon per pool [tC ha^{-1}]. If scalar, initial total soil organic carbon [tC ha^{-1}]. In the latter case, either `model` must be selected or `C0_fracts` must be passed. If `NULL`, filled with zeros.
`N0`	vector with a length equal to the number of pools. Contains initial soil organic nitrogen per pool [tN ha^{-1}]. If `NULL`, filled with zeros. Only used when `calcN = TRUE` and `calcN0 = FALSE`.
`Cin`	either matrix with a number of columns equal to the number of pools and a number of rows corresponding to `t_sim`, or list containing such matrices. If it is a list, each element of the list is expected to represent a stochastic repetition that covers input uncertainties. Then, the list must contain matrices of equal dimensions. Each matrix (or the one if modelling without uncertainties) must contain information about carbon input per pool and time step [tC ha^{-1}]. When `CTool_input_raw = TRUE`, and `model = "C-Tool"` or `model = "C-Tool-org"`, the matrix structure can have two columns (as described for `CTool_input_raw`). If `NULL`, filled with zeros.
`Nin`	either matrix with a number of columns equal to the number of pools and a number of rows corresponding to `t_sim`, or list containing such matrices. If it is a list, each element of the list is expected to represent a stochastic repetition that covers input uncertainties. Then, the list must contain matrices of equal dimensions. Each matrix (or the one if modelling without uncertainties) must contain information about nitrogen input per pool and time step [tN ha^{-1}]. When `CTool_input_raw = TRUE`, and `model = "C-Tool"` or `model = "C-Tool-org"` the matrix structure can have 2 columns (as described for `CTool_input_raw`). If `NULL`, filled with zeros. Must contain entries `> 0` where entries of `Cin` are `> 0`. Only used when `calcN = TRUE`.

`Cin_wood`	list of lengths of different wood diameter classes. Each list element must be in `Cin` format and represent a specific wood diameter. Furthermore, the list elements themselves can be lists and contain stochastic repetitions, as explained for `Cin`. The mean diameter per class is defined in `wood_diam`. Only used when `model = "Yasso15"` or `model = "Yasso20"`.
`Nin_wood`	list of lengths of different wood diameter classes. Each list element must be in `Nin` format and represent a specific wood diameter. Furthermore, the list elements themselves can be lists and contain stochastic repetitions, as explained for `Nin`. The mean diameter per class is defined in `wood_diam`. Must contain entries of `> 0` where entries of `Cin_wood` are `> 0`. Only used when `calcN = TRUE`. Only used when `model = "Yasso15"` or `model = "Yasso20"`.
`wood_diam`	vector with wood diameter [cm]. The first element corresponds to the first list element of `Cin_wood` and `Nin_wood`. If `NULL`, filled with zeros. Only used when `Cin_wood` is specified and when either `model = "Yasso15"` or `model = "Yasso20"`. Must contain entries `>= 0`.
`xi`	either matrix with a number of columns equal to the number of pools and a number of rows corresponding to `t_sim` or list containing such matrices. If it is a list, each element of the list is expected to represent a stochastic repetition that covers input uncertainties. Then, the list must contain matrices of equal dimensions. Each matrix (or the one if modelling without uncertainties) must contain information about time series of rate modifying factors for each model pool, built on the basis of annual decomposition rates. If `NULL`, filled with ones. Only used when `model is NULL`.
`env_in`	matrix with a model-specific number of columns and a number of rows corresponding to `t_sim`. Contains environment-specific information to calculate rate modifying factors (instead of passing them with `xi`) and initial carbon and nitrogen distributions (only RothC). When `model = "RothC"`, it must have four columns: atmospheric temperature (T) [°C], precipitation (p) [mm], evapotranspiration [mm] and a vector of zeros and ones, where ones indicate time steps when the soil is vegetated and zeros when it is bare. When `model = "Yasso07"` or `model = "Yasso15"` or `model = "Yasso20"`, it must have two columns: T [°C] and p [mm]. When `model = "C-Tool"` or `model = "C-Tool-org"`, it has one column: T [°C]. If `NULL`, filled with ones. Only used when `model is not NULL`.
`site`	vector of model-specific length. Contains site-specific information to calculate rate modifying factors (instead of passing them with `xi`) and initial distributions. When `model = "RothC"`, it must be of length four: sample depth [mm], clay content [%], black sand status (0 or 1, 0 if unknown or if black sand method is not desired) and CN ratio (0 if unknown, but then either `C0` and `N0` must be passed or `calcC0 = TRUE` and `calcN0 = TRUE`, information on CN ratio given in `site` always takes precedence over internally calculated CN ratios). When `model = "C-Tool"` or `model = "C-Tool-org"`, it must be of length one: clay content [%]. Only used when `model = "RothC"` or `model = "C-Tool"` or `model = "C-Tool-org"`.

`theta`	either vector with model parameters for predefined models or matrix with rows of such parameters. If it is a matrix, each row is expected to represent a stochastic repetition that covers input uncertainties. If uncertainties are defined by another argument, e.g. `Cin` or `Nin`, these determine the number of stochastic repetitions and not `theta`. Then, if `theta` is a matrix, a parameter vector is randomly drawn for each uncertainty loop. Each vector (or row of matrix) must be of length 7 when `model = "RothC"`, of length 10 when `model = "C-Tool"` or `model = "C-Tool-org"`, of length 21 when `model = "Yasso07"` and of length 30 when `model = "Yasso15"` or `model = "Yasso20"`. If `NULL`, model-specific standard parameters are used instead. Only used when `model is not NULL`. See section 3.3 for details and standard parameters used.
`theta_unc`	either number or vector of percentage values. If it is a vector, the same model-specific lengths as described for `theta` must be used. When used, model parameters modified by taking from the normal distribution around given values (either from `theta` or predefined values) with a standard deviation of `theta_unc`. This will be repeated as many times as defined in `theta_n_unc` or as defined by uncertainty dimensions of a carbon or nitrogen input argument (e.g. `Cin`) and lead to unique model results and output list elements. Only used when `model is not NULL` and `theta` is not a matrix.
`theta_n_unc`	number of stochastic repetitions when model parameters for predefined models should be determined from a random distribution. Only used when the number of stochastic repetitions is not defined by another argument (e.g. `Cin`). Only used when `model is not NULL`, `theta_unc is not NULL` and `theta` is not a matrix.
`meas_data`	matrix with a number of rows equal to the number of measurement points. The first column defines the time of measurement, the metric of which is based on simulation time steps. The second row must contain values of measured soil organic carbon stock. The third row must contain values of measured soil organic nitrogen and is only used when `calcN0 = TRUE`. Only used when `calcC0 = TRUE`.
`t_sim_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `t_sim` format. Only used when `multisite = TRUE`.
`A_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `A` format. Only used when `multisite = TRUE` and `model is NULL`. When `multisite = TRUE`, `A` can be passed instead of `A_sl` to have the same argument for all sites.
`C0_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `C0` format. Only used when `multisite = TRUE`.
`N0_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `N0` format. Only used when `multisite = TRUE`, `calcN = TRUE` and `calcN0 = FALSE`.

`Cin_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `Cin` format, which can also contain uncertainties. Thus, `Cin_sl` can either be a list of different sites each containing lists of different uncertainty representations each with matrices of carbon input as described for `Cin`, or it can simply be a list of different sites each containing such matrices. Only used when `multisite = TRUE`.
`Nin_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `Nin` format, which can also contain uncertainties. Thus, `Nin_sl` can either be a list of different sites each containing lists of different uncertainty representations each with matrices of carbon input as described for `Nin`, or it can simply be a list of different sites each containing such matrices. Only used when `multisite = TRUE`. Must contain entries `> 0` where entries of `Cin_sl` are `> 0`.
`Cin_wood_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `Cin_wood` format, which can also contain uncertainties. Thus, `Cin_wood_sl` can either be a list of different sites each containing lists of different wood diameter representations, which in turn each contain lists of different uncertainty representations, each with matrices of carbon input as described for `Cin`, or it can simply be a list of different sites each containing lists of different wood diameter representations each containing such matrices. Only used when `multisite = TRUE` and either `model = "Yasso15"` or `model = "Yasso20"`.
`Nin_wood_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `Nin_wood` format, which can also contain uncertainties. Thus, `Nin_wood_sl` can either be a list of different sites each containing lists of different wood diameter representations, which in turn each contain lists of different uncertainty representations, each with matrices of carbon input as described for `Nin`, or it can simply be a list of different sites each containing lists of different wood diameter representations each containing such matrices. Only used when `multisite = TRUE` and either `model = "Yasso15"` or `model = "Yasso20"`. Must contain entries `> 0` where entries of `Cin_wood_sl` are `> 0`.
`wood_diam_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `wood_diam` format. Only used when `multisite = TRUE` and when either `model = "Yasso15"` or `model = "Yasso20"`.
`xi_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `xi` format, which can also contain uncertainties. In the latter case, the site list must include uncertainty lists. Only used when `multisite = TRUE` and `model is NULL`.
`env_in_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `env_in` format. Only used when `multisite = TRUE`.
`site_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `site` format. Only used when `multisite = TRUE`.
`sitelist`	list with names of sites to simulate. Only used when `multisite = TRUE`.

`meas_data_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `meas_data` format. Consequently, it is only used when `calcC0 = TRUE` and the third row of each list element is only used when `calcN0 = TRUE`. Only used when `multisite = TRUE`.
`calcN`	logical indicating whether soil organic nitrogen should be modeled.
`calcNbalance`	logical indicating whether the balance of nitrogen cycling should be calculated.
`calcN0`	logical indicating whether the initial amount of nitrogen should be calculated (instead of passing `N0`). Then, the information in `meas_data` is used to determine initial states using linear regression.
`calcC0`	logical indicating whether the initial amount of carbon should be calculated (instead of passing `C0`). Then, the information in `meas_data` is used to determine initial states using linear regression.
`calcCN_fast_init`	logical indicating whether to calculate the initial CN ratio for fast pools (using `Cin` and `Nin`) or whether `CN_fast_init` should be used.
`CTool_input_raw`	logical defining of which type `Cin` and `Nin` are when modelling with `C-Tool`. If `true`, `Cin` and `Nin` can only have two columns, one for the topsoil and one for the subsoil, and `SORCERING` is applying the `C-Tool`-specific distribution to model pools. If `false` (default) `Cin` and `Nin` must have six columns, one per model pool. Only used when `model = "C-Tool"` or `model = "C-Tool-org"`.
`RothC_Cin4C0`	logical defining whether the SOC input should be used for the calculation of initial SOC. If `false` the standard RothC ratio for agricultural soils of DPM to RPM of 0.59 to 0.41 is used. Only used when `model = "RothC"` and `calcC0 = TRUE`.
`C0_fracts`	numerical vector of a length equal to the number of pools. Contains initial fractions of SOC in pools, the sum of which must be 1. Only used when `calcC0 = TRUE`.
`multisite`	logical indicating whether multiple sites should be calculated with one program call. Then, `t_sim`, `C0`, `N0`, `Cin`, `Nin`, `Cin_wood`, `Nin_wood`, `wood_diam`, `env_in`, `site_in`, `xi` and `meas_data` must be of list type and replaced with `t_sim_sl`, `C0_sl`, `N0_sl`, `Cin_sl`, `Nin_sl`, `Cin_wood_sl`, `Nin_wood_sl`, `wood_diam_sl`, `env_in_sl`, `site_in_sl`, `xi_sl` and `meas_data_sl`, respectively. `A`, `CN_bio` and `CN_fast_init` can be given as single variables or in list form of `A_sl`, `CN_bio_sl` and `CN_fast_init_sl`, respectively.
`pooltypes`	integer vector with a length equal to the number of pools. Contains information necessary for the calculation of `N0`. Allowed values are 1-6. 1: topsoil fast pool, 2: topsoil bio or humus pool, 3: topsoil chemically stable or inert pool, 4: subsoil fast pool, 5: subsoil bio or humus pool, 6: subsoil chemically stable or inert pool. Predefined values are (1,1,2,2,3) when `model = "RothC"`, (1,2,3,4,5,6) when `model = "C-Tool"` or `model = "C-Tool-org"`, (1,1,1,2,3) when `model = "Yasso07"` or `model = "Yasso15"` or `model = "Yasso20"`. Only used when `calcN = TRUE` and `calcN0 = TRUE`.

`CN_fast_init`	number that defines the initial CN ratio for fast pools (`pooltypes = 1 or 4`). Only used when `Nin` (or `Nin_sl`) and `Cin` (or `Cin_sl`) do not provide enough information for the estimation of initial nitrogen. The user will be informed about it when `init_info = TRUE`. Only used when `calcN = TRUE` and `calcN0 = TRUE`.
`CN_bio`	number that defines the initial CN ratio for slow pools (`pooltypes = 2 or 5`). Only used when `calcN = TRUE` and `calcN0 = TRUE`.
`CN_fast_init_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `CN_fast_init` format. Only used when `calcN = TRUE`, `multisite = TRUE` and `calcN0 = TRUE`. When `multisite = TRUE`, `CN_fast_init` can be passed instead of `CN_fast_init_sl` to have the same argument for all sites.
`CN_bio_sl`	list with a length of number of sites to simulate. Each list element represents a site and must be in `CN_bio` format. Only used when `calcN = TRUE`, `multisite = TRUE` and `calcN0 = TRUE`. When `multisite = TRUE`, `CN_bio` can be passed instead of `CN_bio_sl` to have the same argument for all sites.
`init_info`	logical indicating whether additional information about the calculation of initial carbon, initial nitrogen, and CN ratio should be printed out during the simulations. Only used when `calcC0 = TRUE` or `calcN0 = TRUE`.
`model`	character string specifying a predefined soil organic carbon model to use. Valid options are `"Yasso07"`, `"Yasso15"`, `"Yasso20"`, `"RothC"`, `"C-Tool"` or `"C-Tool-org"`. When `not NULL`, `xi` and `A` are calculated by `SORCERING`, and `env_in` must be specified. Additionally, `theta` can be specified to not use standard model parameters. (see section 3.3). If `calcN0 = TRUE` and `pooltypes = NULL` model-specific standard values for `pooltypes` are used.

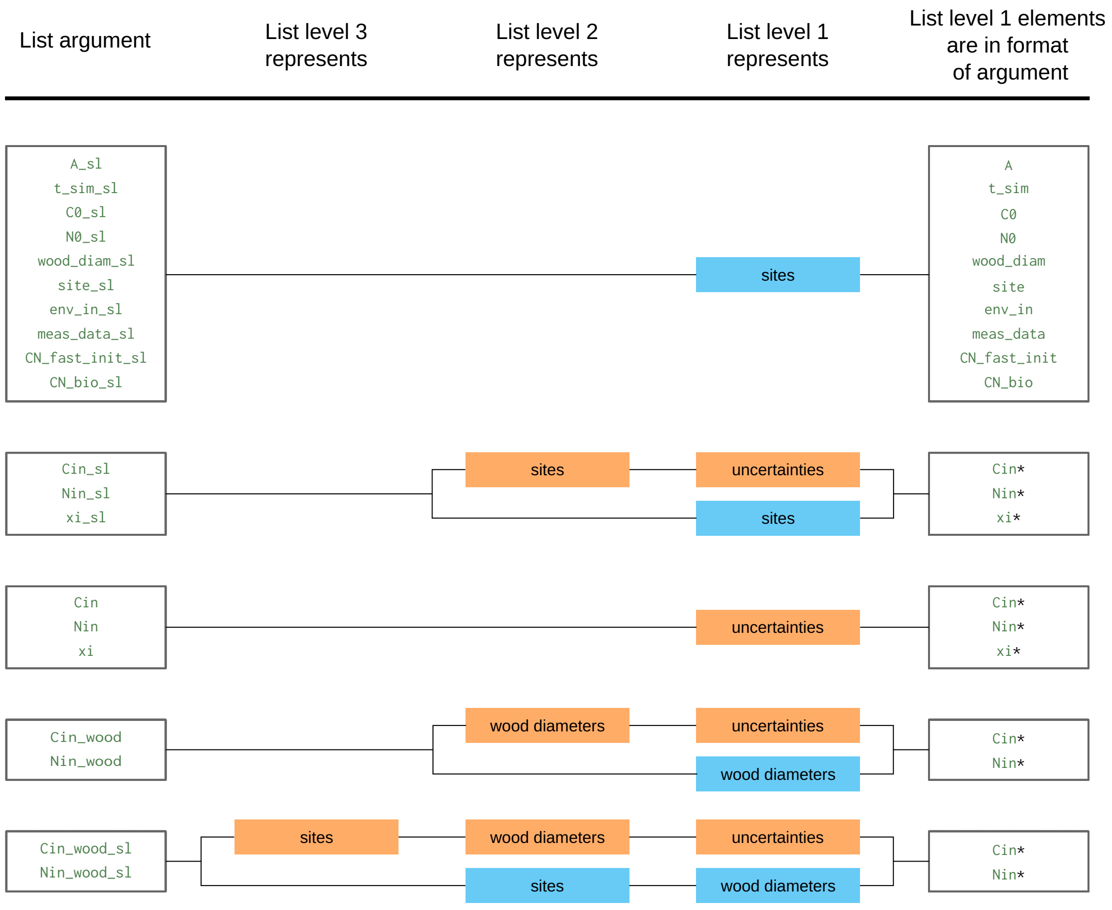

Figure 1: List structure of SORCERING arguments. The arguments listed can either include uncertainties (orange structure path) or not (blue structure path). Arguments marked with an asterisk refer to those in matrix format.

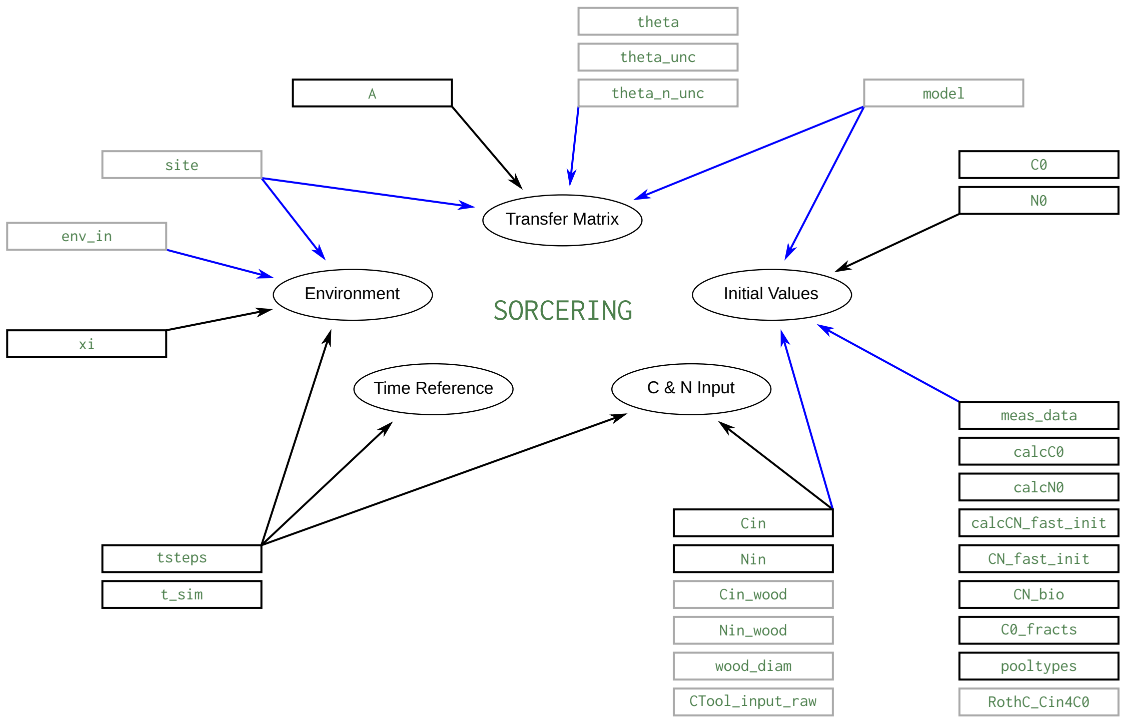

Figure 2: Content structure of SORCERING arguments. Arguments with grey frames depend on using predefined models. Black arrows indicate direct definitions or direct influences. Blue arrows indicate indirect influences that are based on internal function calculations or definitions and can serve as an alternative to the direct definitions.

Value

sorcering() returns either a list of carbon and nitrogen output values or, when multisite = TRUE, a list broken down by site with result lists for each site. When uncertainties are modeled (as can be defined by passing e.g. Cin, Nin, xi or theta), the output is even extended to include another list dimension that covers these uncertainties. The lowest output list-level contains the following components:

C	matrix with a number of rows corresponding to t_sim and a number of columns equal to the number of pools. Contains soil organic carbon [tC ha^{-1}].
N	matrix with a number of rows corresponding to t_sim and a number of columns equal to the number of pools. Contains soil organic nitrogen [tN ha^{-1}]. Only generated when calcN = TRUE
Nloss	matrix with a number of rows corresponding to t_sim and a number of columns equal to the number of pools. Contains nitrogen losses [tN ha^{-1}]. Positive values indicate that nitrogen was lost in the pools between this and the previous time steps (taking nitrogen decomposition and input into account). Only generated when calcN = TRUE.
Nmin	matrix with a number of rows corresponding to t_sim and a number of columns equal to the number of pools. Contains nitrogen mineralisation [tN ha^{-1}]. If values are negative, nitrogen immobilisation exceeds mineralisation. Only generated when calcN = TRUE.
Nmin.sink.1, ..., Nmin.sink.n	matrices with a number of rows corresponding to t_sim and a number of columns equal to the number of pools n. Contain pool-specific nitrogen mineralisation sinks [tN ha^{-1}] (from the pool according to variable index [1, ..., n] to the pool according to column number). If the sink is the pool itself (index equals column number) the amount of decomposition is recorded. Only generated when calcN = TRUE.
Nbalance	matrix with a number of rows corresponding to t_sim and three columns. Contains information on overall N changes in the soil between two time steps (first column) and information on total system N balance calculated based on total Nloss (second column) and based on total Nmin (third column) [tN ha^{-1}]. Only generated when calcN = TRUE and calcNbalance = TRUE.

Package Building Information

The SORCERING code was written in C++ using the R packages Rcpp (Eddelbuettel et al. 2021a) and RcppArmadillo (Eddelbuettel et al. 2021b).

Details

1. Introduction

SORCERING is a general model framework to describe soil organic carbon (SOC) dynamics and soil organic nitrogen (SON) dynamics based on models of first-order kinetics. It can be applied to any given SOC first-order kinetics model. The approach has already been successfully tested to describe SOC dynamics of Yasso (Tuomi et al. 2009, Viskari et al. 2020; 2022), RothC (Coleman and Jenkinson 1996) and C-Tool (Taghizadeh-Toosi et al. 2014). Moreover, it additionally offers the possibility of modelling N immobilisation and mineralisation by enhancing given SOC models by an additional N module. SORCERING was created using the C++ interface Rcpp (Eddelbuettel et al. 2021a) and can handle multiple sites and multiple stochastic representations with just one function call. This makes SORCERING a computationally efficient SOC and SON modelling tool.

In the following a description of each output value (see section 'Value') is given, details of the optional calculation of initial SOC and SON are presented, and predefined existing models and their functions are described.

2. Output Calculation

2.1. Value `C`

`SORCERING` calculates SOC applying a given SOC model for every simulation time step defined by passing `tsteps` and `t_sim`. SOC models applied here are defined by a number of pools, each characterised by specific decomposition and turnover rates. The underlying equation of first-order kinetics defines the change of SOC concentration in time as:

$$\frac{dC(t)}{dt} = Cin(t) + A_e(t) \cdot C(t) \tag{1}$$

The boundary condition $Cin(t)$ and the initial condition $C(0)$ must be defined beforehand (by passing `Cin` and `C0`, or by calculating with `SORCERING`-specific functions). $A_e(t)$ is composed of transfer matrix A (as passed with `A` or provided by `model`) and the model-specific generated rate modifying factor series $xi(t)$ (as passed with `xi` or calculated for a predefined `model`):

$$\begin{aligned} A_e(t) &= \left(A^T \cdot xi(t)\right)^T \\ &= A \cdot diag(xi(t)) \end{aligned} \tag{2}$$

with superscript ^{T} denoting transposed matrices. Eq. 1 is valid for scalar values of A, C, xi and Cin, as well as for a square matrix A with side length of number of SOC pools n and related one dimensional vectors C, xi and Cin with length n. Each element of C, xi and Cin and each row and column of A thus stands for a specific pool. Off-diagonal elements of A describe SOC fluxes and diagonal elements describe SOC decomposition. Analytical solutions of eq. 1 are exponential functions and can be very complex with A containing many off-diagonals, i.e. SOC transfer between many pool pairs. Therefore, numerical solutions are an efficient way to solve the resulting complex equation system. In `SORCERING`, this equation system is solved by applying the fourth-order Runge-Kutta method:

$$C(t) = C(t-1) + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \tag{3}$$

with

$$\begin{aligned} K_1 =& Cin(t-1) + A \cdot diag(xi(t-1)) \cdot C(t-1) \\ K_2 =& Cin(t-1) + A \cdot diag\left(\frac{xi(t-1)+xi(t)}{2}\right) \cdot \left(C(t-1) + \frac{K_1}{2}\right) \\ K_3 =& Cin(t-1) + A \cdot diag\left(\frac{xi(t-1)+xi(t)}{2}\right) \cdot \left(C(t-1) + \frac{K_2}{2}\right) \\ K_4 =& Cin(t-1) + A \cdot diag(xi(t)) \cdot (C(t-1) + K_3) \end{aligned} \tag{4}$$

2.2. Values `N` and `Nloss`

As an extension to SOC modelling, `SORCERING` allows the modelling of SON coupled to the modelling of SOC. Its implementation is based on the following simplifying assumptions: (1) Nitrogen transfer and turnover rates are equal to carbon decomposition and transfer rates. (2) There is no N limitation in the soil, i.e. mineral N is always available for N immobilisation processes. (3) CN ratios of single pools are only affected by external inputs of C and N. The transfer of organic matter among pools does not affect CN ratios. As for SOC, the development of SON depends on boundary and initial conditions: $Nin(t)$ and $N(0)$ (passed with `Nin` and `N0`, or calculated with `SORCERING`-specific functions).

Given the input-considered SOC losses

$$C_{loss} = C(t-1) + Cin(t-1) - C(t) \tag{5}$$

and the input-considered NOC losses

$$N_{loss} = N(t-1) + Nin(t-1) - N(t) \tag{6}$$

between time points t and $t-1$, and assuming equal C and N decomposition rates

$$\frac{C_{loss}}{C(t)} = \frac{N_{loss}}{N(t)} \tag{7}$$

the amount of SON at each simulation time step is then calculated pool-wise as:

$$N(t) = \frac{N(t-1) + Nin(t-1)}{\left(\frac{C_{loss}}{C(t)} + 1\right)} \tag{8}$$

2.3. Values `Nmin`, `Nmin.sink⟨1⟩`, ... , `Nmin.sink⟨n⟩`

Along with modelling SON, further quantities such as mineralisation rates are determined. In the following mineralisation is mentioned as if it includes mineralisation (positive values) and immobilisation (negative values). In contrast to nitrogen losses (eq. 6), mineralisation rates contain information about sources and sinks of SON. Pool-specific N mineralisation $Nmin.sink\,\langle j\rangle$ and N mineralisation $Nmin$ are related as follows:

$$Nmin_j(t) = \sum_{p=1}^{n} Nmin.sink\,\langle j\rangle_p\,(t) \tag{9}$$

for each simulation time point t, each pool $j = 1,...,n$ and each pool $p = 1,...,n$ and n total pools. Or in other words, the row sum of $Nmin.sink\,\langle j\rangle$ at one simulation time point equals the j^th^ column of $Nmin$ at that time point. Mineralisation rates and sinks are read from a mineralisation rates matrix $Nmin.mat$:

$$Nmin_1(t),...,Nmin_n(t) = \sum_{i=1}^{n} Nmin.mat_{i,1}(t),...,\sum_{i=1}^{n} Nmin.mat_{i,n}(t) \tag{10}$$

$$Nmin.sink\langle j\rangle_1(t),...,Nmin.sink\langle j\rangle_n(t) = Nmin.mat_{j,1}(t),...,Nmin.mat_{j,n}(t) \tag{11}$$

and using the fourth-order Runge-Kutta Method *Nmin.mat* at time point *t* is calculated as follows:

$$Nmin.mat(t) = -\frac{1}{6}(Kn_1 + 2Kn_2 + 2Kn_3 + Kn_4) \tag{12}$$

with

$$\begin{aligned}
Kn_1 &= (A \cdot diag(xi(t-1) \cdot C(t-1)))^T \cdot diag\left(\frac{1}{CN(t)}\right)\\
Kn_2 &= \left(A \cdot diag\left(\frac{xi(t-1)+xi(t)}{2} \cdot \left(C(t-1)+\frac{K_1}{2}\right)\right)\right)^T \cdot diag\left(\frac{1}{CN(t)}\right)\\
Kn_3 &= \left(A \cdot diag\left(\frac{xi(t-1)+xi(t)}{2} \cdot \left(C(t-1)+\frac{K_2}{2}\right)\right)\right)^T \cdot diag\left(\frac{1}{CN(t)}\right)\\
Kn_4 &= (A \cdot diag(xi(t) \cdot (C(t-1)+K_3)))^T \cdot diag\left(\frac{1}{CN(t)}\right)
\end{aligned} \tag{13}$$

and superscript ^{T} denoting transposed matrices and

$$CN(t) = \begin{cases} \frac{C(t)}{N(t)}, & \forall N(t) > 0 \\ n.c., & \text{otherwise} \end{cases} \tag{14}$$

and K_1 - K_3 taken from eq. system (4). Note that $Kn_{1..4}$ are matrices and $K_{1..3}$ are vectors.

As changes in SON must match the sums of all mineralisation paths, the sums over soil pools of `Nloss` and `Nmin`, respectively, must be approximately equal for all simulation time points:

$$\textstyle\sum_{p=1}^{n} Nloss_p(t) \approx \sum_{p=1}^{n} Nmin_p(t) \tag{15}$$

A verification of this relation is given by `Nbalance` (see below).

2.4. Value `Nbalance`

The overall N change between two time steps is calculated as:

$$\Delta N(t) = \sum_{p=1}^{n} N_p(t-1) - \sum_{p=1}^{n} N_p(t) \tag{16}$$

The total system N balance serves as a verification output. Both of the following equations should always give results close to zero:

$$N_{bal1}(t) = \sum_{p=1}^{n} Nin_p(t-1) + \Delta N(t) - \sum_{p=1}^{n} Nloss_p(t) \approx 0 \tag{17}$$

$$N_{bal2}(t) = \sum_{p=1}^{n} Nin_p(t-1) + \Delta N(t) - \sum_{p=1}^{n} Nmin_p(t) \approx 0 \tag{18}$$

$\Delta N(t)$ is saved in the first column, $N_{bal1}(t)$ in the second and $N_{bal2}(t)$ in the third column of `Nbalance`.

3. Additional Features and Input Processing

3.1. Estimation of Initial Soil Organic Carbon

If there is no initial SOC data, total initial SOC $\Sigma C(0)_{est}$ can be estimated if there are more than two SOC measurement points (passed with `meas_data`) and the model is called with `calcC0 = TRUE`. Then, $\Sigma C(0)_{est}$ is the intercept value (at the first time point) of the linear regression line through measured SOC data. In order to derive the initial SOC for each model pool, information about the initial SOC shares among pools $C0_{fracts}$ is required. It can be passed with `C0_fracts`) or, when a predefined SOC model is used, the model's standard fractions are taken (see section 3.3). The initial SOC distribution among the pools is then calculated with:

$$C(0)_{p=1}^{n} = \Sigma C(0)_{est} \cdot C0_{fracts,p} \tag{1}$$

with n being the number of SOC pools.

3.2. Estimation of Initial Soil Organic Nitrogen

If there is no initial SON data, similarly to SOC, total initial SON $\Sigma N(0)_{est}$ can be estimated if there are more than two SON measurement points (passed with `meas_data`) and the model is called with `calcN0 = TRUE`. Likewise, $\Sigma N(0)_{est}$ is the intercept value (at the first time point) of the linear regression line through measured SON data. To estimate the initial pool distribution of SON the model used must be classified into fast, slow and resistant pools (either via `pooltypes` or automatically when a predefined SOC model is used). Then, the initial SON in fast pools is calculated as:

$$N_{fast}(0) = \frac{C_{fast}(0)}{CN_{fast}(0)} \tag{2}$$

If `calcCN_fast_init = FALSE` (default), the initial CN ratio of fast pools $CN_{fast}(0)$ is defined in `CN_fast_init` (40 by default). If `calcCN_fast_init = TRUE`, $CN_{fast}(0)$ is estimated from organic inputs, or more specifically, it is thought to be typically the result of the direct input of organic C and N into these pools. Consequently, this method assumes that there has been similar management or natural input in the past, and the organic input during the simulation is used to estimate the initial CN ratio:

$$CN_{fast}(0) = \sum_{t=t_{in,first}}^{t_{in,last}} CN_{in}(t) \cdot weight(t) \tag{3}$$

However, not each input should have the same impact. Therefore, higher weights for early inputs and for higher amounts of inputs are used:

$$weight(t) = \frac{weight_t(t) \cdot weight_c(t)}{\sum_{T=t_{in,first}}^{t_{in,last}} \left(weight_t(T) \cdot weight_c(T)\right)} \tag{4}$$

trivially with

$$\sum_{t=t_{in,first}}^{t_{in,last}} weight(t) = 1 \tag{5}$$

and with mass weight defined as:

$$weight_c(t) = \frac{C_{in}(t)}{\sum_{t=t_{in,first}}^{t_{in,last}} C_{in}(t)} \tag{6}$$

and temporal weight defined as:

$$weight_t(t) = \frac{1}{t^2} \tag{7}$$

If the sum of total carbon input during the simulation is smaller than the initial SOC, it is assumed that calculating the CN ratio from the inputs alone would lead to unrealistic results. As a solution, the missing amount of carbon is placed at the beginning of the C_{in} vector and `CN_fast_init` at the beginning of the CNfast vector for the calculations of equations 3 and 6. The user will be informed about those events when `init_info = TRUE`.

The initial amount of SON in slow and resistant pools $N_{slow}(0)$ and $N_{res}(0)$ are calculated as follows:

$$N_{slow}(0) = \frac{C_{slow}(0)}{CN_{slow}(0)} \tag{8}$$

$$N_{res}(0) = \frac{\Sigma C(0)}{\Sigma CN(0)_{est}} - \left(N_{fast}(0) + N_{slow}(0)\right) \tag{9}$$

with CN_{slow} initially starting with the given value of `CN_bio` (9 by default). If unrealistic values result out of the calculation, i.e. $N_{res}(0) < 0$ or $CN_{res}(0) = \frac{C_{res}(0)}{N_{res}(0)} > 100$, $CN_{slow}(0)$ iteratively gets raised by 0.1 and eqs. 8 and 9 are iteratively repeated until realistic results are achieved. Assuming higher CN values in resistant pools than in slow pools, if $CN_{res}(0)$ is smaller than $CN_{slow}(0)$, $CN_{slow}(0)$ iteratively gets lowered by 0.1 and eqs. 8 and 9 are iteratively repeated until $CN_{res}(0)$ is not smaller than $CN_{slow}(0)$. Initial total CN ratio $CN(0)$ is calculated as follows:

$$\Sigma CN(0)_{est} = \frac{\Sigma C(0)_{est}}{\Sigma N(0)_{est}} \tag{10}$$

and $C_i(0)$ and $N_i(0)$ being the intercept values by applying linear regressions through measured SOC and SON values, respectively. One should note that $\Sigma C(0)$ may not be the same as $\Sigma C(0)_{est}$ and therefore use the estimated SOC for the measured data-based calculation of $N(0)$.

3.3. Predefined Models

`SORCERING` can be run using existing predefined SOC models. The information about the transfer matrix and the rate modifying factors, which are formerly passed with `A` and `xi`, are then calculated based on environment-specific information `env_in` (mandatory) and model parameters `theta` (when not passed, standard values will be used). At the moment one can choose among five models: Yasso07 (Tuomi et al. 2009), Yasso15 (Viskari et al. 2020), Yasso20 (Viskari et al. 2022), RothC (Coleman and Jenkinson 1996) or C-Tool (Taghizadeh-Toosi et al. 2014).

According to the definitions given in section 2.1, A is a matrix of size $n \times n$ and $xi(t)$ is a vector of length n, with n being the number of pools. For all predefined models, $n = 5$, except for C-Tool, where $n = 6$. When using predefined models, the $xi(t)$ vectors are calculated based on annual decomposition rates. Using the default value of `tsteps = "monthly"`, $xi(t)$ is divided by 12, which is not taken into account below.

When using predefined SOC models, the `pooltypes` vector does not need to be passed since default values are used. These values were created be the `SORCERING` developers out of logical considerations and with the aim of SON initialization.

3.3.1. Yasso

When modelling with Yasso, the user must pass either `model = "Yasso07"`, `model = "Yasso15"` or `model = "Yasso20"` according to the three versions provided.

Parameters

When `model = "Yasso07"`, a `theta` default vector of

$$\Theta_{Y07} = (kA, kW, kE, kN, kH, p_1 ... p_{12}, pH, \beta_1, \beta_2, \gamma) \tag{11}$$

is used, and when `model = "Yasso15"` or `model = "Yasso20"`, a `theta` default vector of

$$\Theta_{Y15} = \Theta_{Y20} = (kA, kW, kE, kN, kH, p_1 ... p_{12}, pH, \beta_1, \beta_2, \beta_{N1}, \beta_{N2}, \beta_{H1}, \beta_{H2}, \gamma, \gamma_N, \gamma_H, \theta_1, \theta_2, r) \tag{12}$$

is used, with standard values listed in Table 1.

Table 1: Yasso standard parameters.

	Yasso07	Yasso15	Yasso20	
kA	0.66	0.49	0.51	Base decomposition rate for pool A [yr^{-1}]
kW	4.3	4.9	5.19	Base decomposition rate for pool W [yr^{-1}]
kE	0.35	0.25	0.13	Base decomposition rate for pool E [yr^{-1}]
kN	0.22	0.095	0.1	Base decomposition rate for pool N [yr^{-1}]
kH	0.0033	0.0013	0.0015	Base decomposition rate for pool H [yr^{-1}]
p_1	0.32	0.44	0.5	Transference fraction from pool A to pool W
p_2	0.01	0.25	0	Transference fraction from pool A to pool E
p_3	0.93	0.92	1	Transference fraction from pool A to pool N
p_4	0.34	0.99	1	Transference fraction from pool W to pool A
p_5	0	0.084	0.99	Transference fraction from pool W to pool E
p_6	0	0.011	0	Transference fraction from pool W to pool N
p_7	0	0.00061	0	Transference fraction from pool E to pool A
p_8	0	0.00048	0	Transference fraction from pool E to pool W
p_9	0.01	0.066	0	Transference fraction from pool E to pool N
p_{10}	0	0.00077	0	Transference fraction from pool N to pool A
p_{11}	0	0.1	0.163	Transference fraction from pool N to pool W
p_{12}	0.02	0.65	0	Transference fraction from pool N to pool E
pH	0.04	0.0046	0.0015	Transference fraction from AWEN pools to pool H
β_1	0.076	0.091	0.158	First-order temperature parameter for AWE pools [°C^{-1}]
β_2	-0.00089	-0.00021	-0.002	Second-order temperature parameter for AWE pools [°C^{-2}]
β_{N1}	-	0.049	0.17	First-order temperature parameter for N pool [°C^{-1}]
β_{N2}	-	-0.000079	-0.005	Second-order temperature parameter for N pool [°C^{-2}]
β_{N1}	-	0.035	0.067	First-order temperature parameter for H pool [°C^{-1}]
β_{H2}	-	-0.00021	0	Second-order temperature parameter for H pool [°C^{-2}]
γ	-1.27	-1.8	-1.44	Precipitation impact parameter for AWE pools [yr mm^{-1}]
γ_N	-	-1.2	-2	Precipitation impact parameter for N pool [yr mm^{-1}]
γ_H	-	-13	-6.9	Precipitation impact parameter for H pool [yr mm^{-1}]
θ_1	-	-0.44	-2.55	First-order impact parameter for wood size [cm^{-1}]
θ_2	-	1.3	1.24	Second-order impact parameter for wood size [cm^{-2}]
r	-	0.26	0.25	Exponent parameter for wood size

Transfer Matrix

For all Yasso versions A looks like this:

$$A = \begin{pmatrix} -kA & p_1 & p_2 & p_3 & 0 \\ p_4 & -kW & p_5 & p_6 & 0 \\ p_7 & p_8 & -kE & p_9 & 0 \\ p_{10} & p_{11} & p_{12} & -kN & 0 \\ pH & pH & pH & pH & -kH \end{pmatrix} \tag{13}$$

Environmental Factors

When `model = "Yasso07"`, $xi(t)$ is calculated as follows:

$$\begin{aligned} xi_1(t) =& xi_2(t) = xi_3(t) = xi_4(t) = xi_5(t) \\ =& e^{\beta_1 T + \beta_2 T^2} \left(1 - e^{\gamma p}\right) \end{aligned} \tag{14}$$

When `model = "Yasso15"` or `model = "Yasso20"`, $xi(t)$ is calculated as follows:

$$\begin{aligned} xi_1(t) =& xi_2(t) = xi_3(t) \\ =& e^{\beta_1 T(t) + \beta_2 T(t)^2} \left(1 - e^{\gamma p(t)}\right) \end{aligned} \tag{15}$$

$$xi_4(t) = e^{\beta_{N1} T(t) + \beta_{N2} T(t)^2} \left(1 - e^{\gamma_N p(t)}\right) \tag{16}$$

$$xi_5(t) = e^{\beta_{H1} T(t) + \beta_{H2} T(t)^2} \left(1 - e^{\gamma_H p(t)}\right) \tag{17}$$

with T in degrees Celsius and p in mm, both passed with `env_in`.

Initial Pool Properties

Initial pool properties described here apply equally to all three Yasso versions. When `calcN0 = TRUE`, a `pooltypes` default vector of $(1,1,1,2,3)$ is used. When `calcC0 = TRUE` and `C0_fracts = Null`, a `C0_fracts` default vector of $(0.15, 0.025, 0.025, 0.35, 0.45)$ is used, which could be a plausible initial carbon distribution for the Yasso pools. However, these values are not intended by the Yasso developers. It is recommended to perform model spinup runs for the initialisation of SOC.

Input from Woody Litter

Woody litter is only considered when `model = "Yasso15"` or `model = "Yasso20"`. Then instead of `Cin` or `Cin_sl`, and instead of `Nin` or `Nin_sl`, `Cin_wood` or `Cin_wood_sl`, and `Nin_wood` or `Nin_wood_sl` must be passed. When `model = "Yasso15"` or `model = "Yasso20"`, by default an additional set of rate modifying factors $xi_w(t)$ for each wood diameter class is calculated:

$$xi_w(t) = min(1, \left(1 + \theta_1 \cdot diam + \theta_2 \cdot diam^2\right)^{-r}) \tag{18}$$

with θ_1,θ_2 and r as described in section 3.3.1 and *diam* being the wood diameter as described in `wood_diam` or `wood_diam_sl`. Equation 2 then is exchanged with:

$$\begin{aligned} A_e(t) &= \left(A^T \cdot xi(t) \cdot xi_w(t)\right)^T \\ &= A \cdot diag(xi(t) \cdot xi_w(t)) \end{aligned} \tag{19}$$

Since each wood diameter produces a new $xi_w(t)$ the model calculation for each wood class is automatically run separately and after each time step the overall new SOC and SON are calculated as weighted means over all wood diameter input runs. The idea behind this separation of input diameter classes is that woody components are subject to delayed decomposition.

3.3.2. RothC

The RothC application used here uses standard RothC parameters and properties for agricultural soils (Coleman and Jenkinson 2014). This particularly affects the estimation of initial SOC. Therefore, especially if modelling non-agricultural soils, it is recommended to use the SOC input and pass `RothC_Cin4C0 = TRUE` to estimate the initial SOC.

Parameters

When `model = "RothC"`, a `theta` default vector of

$$\Theta_R = (k_{DPM}, k_{RPM}, k_{BIO}, k_{HUM}, k_{IOM}, R_{W,max}, R_{W,min}) \tag{20}$$

is used, with standard values listed in Table 2. Note that $R_{W,max}$ and $R_{W,min}$ are not typically intended by RothC developers to be modifiable parameters.

Table 2: RothC standard parameters.

k_{DPM}	10	Decomposition rate for DPM pool [yr^{-1}]
k_{RPM}	0.3	Decomposition rate for RPM pool [yr^{-1}]
k_{BIO}	0.66	Decomposition rate for BIO pool [yr^{-1}]
k_{HUM}	0.02	Decomposition rate for HUM pool [yr^{-1}]
k_{IOM}	0	Decomposition rate for IOM pool [yr^{-1}]
$R_{W,max}$	1	Maximum rate modifying factor for soil moisture
$R_{W,min}$	0.2	Minimum rate modifying factor for soil moisture

Transfer Matrix

A is build as follows:

$$A = \begin{pmatrix} -k_{DPM} & 0 & 0 & 0 & 0 \\ 0 & -k_{RPM} & 0 & 0 & 0 \\ B \cdot k_{DPM} & B \cdot k_{RPM} & B \cdot k_{BIO} - k_{BIO} & B \cdot k_{HUM} & 0 \\ H \cdot k_{DPM} & H \cdot k_{RPM} & H \cdot k_{BIO} & H \cdot k_{HUM} - k_{HUM} & 0 \\ 0 & 0 & 0 & 0 & -k_{IOM} \end{pmatrix} \tag{21}$$

with

$$B = \frac{0.46}{1.67\,(1.85 + 1.6e^{-0.0786 \cdot clay})} \tag{22}$$

$$H = \frac{0.54}{1.67\,(1.85 + 1.6e^{-0.0786 \cdot clay})} \tag{23}$$

and *clay* as defined in `site`.

Environmental Factors

$xi(t)$ is calculated as follows:

$$\begin{aligned} xi_1(t) =& xi_2(t) = xi_3(t) = xi_4(t) = xi_5(t) \\ =& R_T(t) \cdot R_C(t) \cdot R_W(t) \end{aligned} \tag{24}$$

with $R_T(t)$ being the rate modifying factor series for temperature:

$$R_T(t) = \begin{cases} \frac{47.91}{1+e^{\left(\frac{106.06}{T(t)+18.27}\right)}}, & \forall T(t) > -18.27 \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

and $R_C(t)$ being the soil cover rate modifying factor:

$$R_C(t) = \begin{cases} 1, & \forall \text{ bare soils} \\ 0.6, & \forall \text{ vegetated soils} \end{cases} \tag{26}$$

and $R_W(t)$ being the topsoil moisture deficit (TSMD) rate modifying factor:

$$R_W(t) = \begin{cases} R_{W,max}, & \forall TSMD_{acc}(t) \geq 0.444 \cdot TSMD_{max}(t) \\ R_{W,min} + (R_{W,max} - R_{W,min}) \frac{TSMD_{max}(t) - TSMD_{acc}(t)}{TSMD_{max}(t) - 0.444 \cdot TSMD_{max}(t)}, & \text{otherwise} \end{cases} \tag{27}$$

and $TSMD_{max}(t)$ being the maximum TSMD

$$TSMD_{max}(t) = \frac{-(20 + 1.3 \cdot clay - 0.01 \cdot clay^2)\frac{depth}{23}}{crp} \tag{28}$$

and

$$crp = \begin{cases} 1.8, & \forall \text{ bare soils} \\ 1, & \forall \text{ vegetated soils} \end{cases} \tag{29}$$

and $TSMD_{acc}(t)$ being the accumulated TSMD

$$TSMD_{acc}(t) = \begin{cases} 0, & \forall p(T) > ETP(t)\, and\, t = 0 \\ p(0) - ETP(0), & \forall p(t) \leq ETP(t)\, and\, t = 0 \\ max(TSMD_{max}(t), R_W(t-1) + p(t) - ETP(t)), & \forall R_W(t-1) < p(t) - ETP(t)\, and\, t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{30}$$

with $clay$ and $depth$ as defined in `site` and T, p, ETP and soil covers as defined in `env_in`.

Initial Pool Properties

When `calcN0 = TRUE`, a `pooltypes` default vector of $(1,1,2,2,3)$ is used. When `calcC0 = TRUE` and `C0_fracts = Null`, initial SOC fractions are calculated for the first four SOC pools, and the initial SOC of the fifth pool $C(0)_{IOM}$ (inert pool IOM, as described by Falloon et al. 1998) is calculated directly out of the total $C(0)$:

$$C(0)_{IOM} = 10^{-1.31 + 1.139 \cdot log_{10}(C(0) - C0_{rel})} + C0_{rel} \tag{31}$$

with $C0_{rel}$ increasing the SOC in the inert pool of black sands based on a method of Springob and Kirchmann (2010):

$$C0_{rel} = \begin{cases} 0, & \forall CN(0) \leq 11 \vee bs_{flag} = 0 \\ \frac{C(0)(11 - CN(0))}{CN(0)\left(\frac{11}{35} - 1\right)}, & \forall CN(0) > 11 \wedge bs_{flag} = 1 \end{cases} \tag{32}$$

and bs_{flag} being the identifier of black sand soils as defined in `site`, and $CN(0)$ being the CN ratio as defined in `site` or, if specified there by entry `= 0`, as calculated. In the letter case, the CN ratio is calculated either as in eq. 10 or from $C0$ and $N0$, depending on whether measured data should be used to estimate the initial CN ratio.

The initial state of the remaining carbon pools is calculated using a simplified version of the analytical solution from Dechow et al. (2019) assuming steady-state conditions. The initial SOC distribution among the five pools is then calculated as:

$$(C(0)_1, C(0)_2, C(0)_3, C(0)_4, C(0)_5) = (f_1 \cdot C(0), f_2 \cdot C(0), f_3 \cdot C(0), f_4 \cdot C(0), C(0)_{IOM}) \tag{33}$$

with

$$(f_1, f_2, f_3, f_4) = (f_{dpm}, f_{rpm}, f_{bio}, f_{hum}) \cdot \frac{C(0) - C(0)_{IOM}}{C(0)} \tag{34}$$

and

$$f_{dpm} + f_{rpm} + f_{bio} + f_{hum} = 1 \tag{35}$$

Following Dechow et al. (2019), these fractions are calculated as:

$$\begin{aligned} f_{dpm} &= \frac{fractI_1 \cdot k_{DPM}^{-1}}{fractI_1 \cdot u_{dpm} + fractI_2 \cdot u_{rpm}} \\ f_{rpm} &= \frac{fractI_2 \cdot k_{RPM}^{-1}}{fractI_1 \cdot u_{dpm} + fractI_2 \cdot u_{rpm}} \\ f_{bio} &= \frac{fractI_1 \cdot u_{bio,dpm} + fractI_2 \cdot u_{bio,rpm}}{fractI_1 \cdot u_{dpm} + fractI_2 \cdot u_{rpm}} \\ f_{hum} &= \frac{fractI_1 \cdot u_{hum,dpm} + fractI_2 \cdot u_{hum,rpm}}{fractI_1 \cdot u_{dpm} + fractI_2 \cdot u_{rpm}} \end{aligned} \tag{36}$$

with

$$fractI_1 = \begin{cases} 0.59, & \texttt{RothC\_Cin4C0 = FALSE} \\ \frac{C_{in,tot,DPM}}{C_{in,tot,DPM} + C_{in,tot,RPM}}, & \texttt{RothC\_Cin4C0 = TRUE} \end{cases} \tag{37}$$

and

$$fractI_2 = 1 - fractI_1 \tag{38}$$

and $C_{in,tot,DPM}$ and $C_{in,tot,RPM}$ being the total SOC inputs (during the following simulation) into DPM and RPM pools, and

$$\begin{aligned} u_{dpm} &= k_{DPM}^{-1} + u_{bio,dpm} + u_{hum,dpm} \\ u_{rpm} &= k_{RPM}^{-1} + u_{bio,rpm} + u_{hum,rpm} \end{aligned} \tag{39}$$

and

$$u_{hum,dpm} = u_{hum,rpm} = \frac{1}{a_{1,2}} (-u_{bio,dpm}) \cdot a_{1,1} - \alpha_1 \tag{40}$$

and

$$u_{bio,dpm} = u_{bio,rpm} = \frac{\alpha_2 \cdot a_{1,2} - \alpha_1 \cdot a_{2,2}}{a_{1,1} \cdot a_{2,2} - a_{1,2} \cdot a_{2,1}} \tag{41}$$

and

$$\begin{aligned} a_{1,1} &= k_{BIO} \cdot (\alpha_1 - 1) \\ a_{1,2} &= k_{HUM} \cdot \alpha_1 \\ a_{2,1} &= k_{BIO} \cdot \alpha_2 \\ a_{2,2} &= k_{HUM} \cdot (\alpha_2 - 1) \end{aligned} \tag{42}$$

and

$$\begin{aligned} \alpha_1 &= cue \cdot 0.46 \\ \alpha_2 &= cue \cdot 0.54 \end{aligned} \tag{43}$$

and carbon use efficiency

$$cue = \frac{1}{1 + 1.67(1.85 + 1.6e^{-0.0786 \cdot clay})} \tag{44}$$

and *clay* being the clay content as defined in `site`.

3.3.3. C-Tool

C-Tool typically needs to be run twice, one time with `model = "C-Tool"` with `Cin` (and if desired `Nin`) of crop input and one time with `model = "C-Tool-org"` with `Cin` (and `Nin`) of organic input. In case of `model = "C-Tool-org"`, `C0` (and `N0`) should be zero. The user must calculate the overall C-Tool result independently as the sum of both runs. Thus, `"C-Tool-org"` is not a separate model, but rather a running mode of C-Tool.

Parameters

When `model = "C-Tool"` or `model = "C-Tool-org"`, a `theta` default vector of

$$\Theta_C = (k_{FOMt}, k_{HUMt}, k_{ROMt}, k_{FOMs}, k_{HUMs}, k_{ROMs}, t_f, f_{CO_2}, f_{ROM}, f_{HUM}) \tag{45}$$

is used, with standard values taken from Taghizadeh-Toosi (2015) listed in Table 3.

Table 3: C-Tool standard parameters.

	C-Tool	C-Tool-org	
k_{FOMt}	1.44	1.44	Decomposition rate for FOM pool (topsoil) [yr^{-1}]
k_{HUMt}	0.0336	0.0336	Decomposition rate for HUM pool (topsoil) [yr^{-1}]
k_{ROMt}	0.000463	0	Decomposition rate for ROM pool (topsoil) [yr^{-1}]
k_{FOMs}	1.44	1.44	Decomposition rate for FOM pool (subsoil) [yr^{-1}]
k_{HUMs}	0.0336	0.0336	Decomposition rate for HUM pool (subsoil) [yr^{-1}]
k_{ROMs}	0.000463	0	Decomposition rate for ROM pool (subsoil) [yr^{-1}]
t_f	0.03	0	Fraction going to downward transport
f_{CO_2}	0.628	0.628	Fraction of CO$_2$ released
f_{ROM}	0.012	0	Fraction of fresh organic matter going to ROM pool
f_{HUM}	0	0.358	Fraction of input going to HUM pool

Transfer Matrix

A is build as follows:

$$A = \begin{pmatrix} -k_{FOMt} & 0 & 0 & 0 & 0 & 0 \\ ai_{21} & -k_{HUMt} & 0 & 0 & 0 & 0 \\ 0 & ai_{32} & -k_{ROMt} & 0 & 0 & 0 \\ ai_{41} & 0 & 0 & -k_{FOMs}+ai_{44} & 0 & 0 \\ 0 & ai_{52} & 0 & ai_{54} & -k_{HUMs}+ai_{55} & 0 \\ 0 & 0 & ai_{63} & 0 & ai_{65} & -k_{ROMs}+ai_{66} \end{pmatrix} \tag{46}$$

with

$$\begin{aligned} ai_{21} &= (1-t_f)\cdot h\cdot k_{FOMt} \\ ai_{41} &= t_f \cdot k_{FOMt} \\ ai_{32} &= f_{ROM}\cdot k_{HUMt} \\ ai_{52} &= (1-f_{CO_2}-f_{ROM})\cdot k_{HUMt} \\ ai_{63} &= (1-f_{CO_2})\cdot k_{ROMt} \\ ai_{44} &= t_f\cdot k_{FOMs} \\ ai_{54} &= (1-t_f)\cdot h\cdot k_{FOMs} \\ ai_{65} &= f_{ROM}\cdot k_{HUMs} \\ ai_{55} &= (1-f_{CO_2}-f_{ROM})\cdot k_{HUMs} \\ ai_{66} &= (1-f_{CO_2})\cdot k_{ROMs} \end{aligned} \tag{47}$$

and *h* being a factor that depends on clay content:

$$h = \frac{1}{1+1.67(1.85+1.6\cdot e^{-7.86\frac{clay}{100}})} \tag{48}$$

and *clay* being the clay content as defined in `site`.

Input Transformation

If the input of SOC and SON is not defined for the single pools but only for topsoil and subsoil fractions (`CTool_input_raw = TRUE`) the standard way of distributing the fractions among the pools is used:

$$\begin{aligned} &Cin(t) = \\ &(Cin_{raw,top} \cdot (1-F_{hum,clay}), Cin_{raw,top} \cdot F_{hum,clay}, 0, Cin_{raw,sub} \cdot (1-F_{hum,clay}), Cin_{raw,sub} \cdot F_{hum,clay}, 0) \\ &Nin(t) = \\ &(Nin_{raw,top} \cdot (1-F_{hum,clay}), Nin_{raw,top} \cdot F_{hum,clay}, 0, Nin_{raw,sub} \cdot (1-F_{hum,clay}), Nin_{raw,sub} \cdot F_{hum,clay}, 0) \end{aligned} \tag{49}$$

with

$$F_{hum,clay} = max(0, f_{HUM} - h) \tag{50}$$

and f_{hum} and h as defined above, and with $Cin_{raw,top}$ and $Nin_{raw,top}$ being topsoil inputs and $Cin_{raw,sub}$ and $Nin_{raw,sub}$ being subsoil inputs.

Environmental Factors

$xi(t)$ is calculated as follows:

$$\begin{aligned} xi_1(t) = xi_2(t) = xi_3(t) = ft_t(t) \\ xi_4(t) = xi_5(t) = xi_6(t) = ft_s(t) \end{aligned} \tag{51}$$

and

$$\begin{aligned} ft_t &= 7.24 \cdot e^{-3.432+0.168 T_{est,t}(t)(1-\frac{T_{est,t}(t)}{73.8}} \\ ft_s &= 7.24 \cdot e^{-3.432+0.168 T_{est,s}(t)(1-\frac{T_{est,s}(t)}{73.8}} \end{aligned} \tag{52}$$

and

$$\begin{aligned} T_{est,t} &= \bar{T} + amp \cdot e^{-\frac{d_t}{10 \cdot d_{damp}}} \cdot sin(\rho \cdot (doy - offset)) \cdot 86400 - \frac{d_t}{10 \cdot d_{damp}} \\ T_{est,s} &= \bar{T} + amp \cdot e^{-\frac{d_s}{10 \cdot d_{damp}}} \cdot sin(\rho \cdot (doy - offset)) \cdot 86400 - \frac{d_s}{10 \cdot d_{damp}} \end{aligned} \tag{53}$$

with $\bar{T}$ being the average annual temperature, amp being the temperature range that lays between the 25% and the 75% quantile, d_t the number of the decimeter-wide soil layer that is relevant for the topsoil, d_s the one that is relevant for the subsoil (with standard values $d_t = 2\,\text{dm}$ and $d_s = 3\,\text{dm}$), furthermore with d_{damp} being the damping depth, ρ being the angular frequency ($= \pi \cdot 2/365/24/3600 = 1.992385 \cdot$

$10^{-7}\,\text{sec}^{-1}$), doy being the day of the year and $offset$ being defined as 110 days. $\bar{T}$ is calculated using data from `tsteps` and `env_in`. Since amp is based on within-year fluctuations it is recommended to use a temporal resolution not coarser than a monthly one. Note that the function from `RcppArmadillo` (Eddelbuettel et al. 2021b) used to calculate the quantiles is equivalent to type 5 of the R `quantile` function. d_{damp} is calculated as follows:

$$d_{damp} = \sqrt{\frac{2Th_{diff}}{\rho}} = 1.874401\,\text{m} \tag{54}$$

with default thermal diffusivity of solid soil Th_{diff} of $0.35 \cdot 10^{-6}\,\text{m}^2\text{sec}^{-1}$.

Initial Pool Properties

When `calcN0 = TRUE`, a `pooltypes` default vector of $(1,2,3,4,5,6)$ is used. When `calcC0 = TRUE` and `C0_fracts = Null`, a `C0_fracts` default vector of

$$\begin{aligned} C0_{fracts} =& (f_{FOMt}, f_{HUMt}, f_{ROMt}, f_{FOMs} \cdot sf, f_{HUMs} \cdot sf, f_{ROMs} \cdot sf) \\ =& (0.0316, 0.4804, 0.488, 0.003 \cdot sf, 0.3123 \cdot sf, 0.6847 \cdot sf) \\ =& (0.0316, 0.4804, 0.488, 0.00338298, 0.35216822, 0.77210880) \end{aligned} \tag{55}$$

is used, with f_{FOMt}, f_{HUMt} and f_{ROMt} being the fresh organic, humus and resistant topsoil matter fractions, and f_{FOMt}, f_{HUMt} and f_{ROMt} being those of the subsoil, and with sf being the subsoil factor defined as

$$sf = \frac{0.53}{0.47} = 1.12766 \tag{56}$$

Note that by default the first three pools in C-Tool represent the upper soil. Thus, to compare with other SOC models only the first three pools of the C-Tool output are relevant.

Examples

```
#1 Example of RothC application with fictional input for a single site

#1.1 Input

data(RothC_Cin_ex, RothC_Nin_ex, RothC_N0_ex, RothC_C0_ex, RothC_xi_ex,
  RothC_site_ex, RothC_env_in_ex) #fictional data

#1.2 Simulations

#In the following two methods are presented, one with a RothC as a predefined
#model (1.2.1), one where the RothC rate modifying factors must be calculated
#beforehand (1.2.2). Both methods lead to the same results.

#1.2.1 Simulation with predefined model

out_rothC <- sorcering( model="RothC", site=RothC_site_ex, env_in=RothC_env_in_ex,
  t_sim=60, Cin=RothC_Cin_ex, Nin=RothC_Nin_ex, N0=RothC_N0_ex, C0=RothC_C0_ex,
  calcN=TRUE, tsteps="monthly")

#1.2.2 Simulation with own model definition and rate modifying factor definition

A_RothC <- fget_A_RothC(clay=30) #create transfer matrix for RothC
out_rothC_own <- sorcering(A=A_RothC , xi=RothC_xi_ex, t_sim=60, Cin=RothC_Cin_ex,
  Nin=RothC_Nin_ex, N0=RothC_N0_ex, C0=RothC_C0_ex, calcN=TRUE, tsteps="monthly")
#Note that RothC_xi_ex contains site and model specific rate modifying factors that
#are only valid in this specific example. Generally, xi must be calculated by the
#user for different environmental conditions and SOC models used.

#1.3 Results

#output structure summary
summary(out_rothC)

#show that results of 1.2.1 and 1.2.2 differ negligibly
all( abs(out_rothC$C-out_rothC_own$C) < 1e-14)
all( abs(out_rothC$N-out_rothC_own$N) < 1e-14)

#example plot
  oldpar <- par(no.readonly = TRUE) #save old par
par(mfrow=c(1,1),mar=c(4,4,1,4))
plot(rowSums(out_rothC$N),axes=FALSE, col=1, cex.lab=2,xlab="",ylab="",ylim=c(0,9),
  pch=20)
par(new=TRUE)
plot(rowSums(RothC_Cin_ex)/rowSums(RothC_Nin_ex),
  axes=FALSE,col=2, cex.lab=2,xlab="",ylab="",ylim=c(0,60),pch=20)
axis(side=2, pos = 0,
  labels = (0:6)*1.5, at=(0:6)*10, hadj=0.7, padj = 0.5, cex.axis=2,las=1,col.axis=1)
axis(side=4, pos = 60,
  labels = (0:6)*10, at=(0:6)*10, hadj=0, padj = 0.5, cex.axis=2, las=1,col.axis=2)
axis(side=1, pos = 0,
  labels =  (0:6)*10 , at=(0:6)*10, hadj=0.5, padj = 0, cex.axis=2)
title(ylab=expression("total N  [t ha"^-1*"]"), line=2, cex.lab=2)
mtext("C input / N input", side=4, line=2, cex=2,col=2)
title(xlab="time", line= 2, cex.lab=2)
par(oldpar) #back to old par
```

```
#2 Example of RothC application with fictional input for a multiple site application

#2.1 Input

data(RothC_Cin_ex_sl, RothC_Nin_ex_sl, RothC_N0_ex, RothC_C0_ex, RothC_site_ex,
  RothC_env_in_ex) #fictional data

#2.2. Simulation

out_multi_rothC <- sorcering( model="RothC", site_sl=rep(list(RothC_site_ex),3),
  env_in_sl=rep(list(RothC_env_in_ex),3), t_sim_sl=list(60,60,60),
  Cin_sl=RothC_Cin_ex_sl, Nin_sl=RothC_Nin_ex_sl, N0_sl=rep(list(RothC_N0_ex),3),
  C0_sl=rep(list(RothC_C0_ex),3), calcN=TRUE, tsteps="monthly", multisite=TRUE,
  sitelist=list("normal","half_input","double_Cin"))

#2.3 Results

#output structure summary
summary(out_multi_rothC$normal)
summary(out_multi_rothC$half_input)
summary(out_multi_rothC$double_Cin)

#example plot
oldpar <- par(no.readonly = TRUE) #save old par
par(mfrow=c(1,1),mar=c(4,4,1,4))
for (listelement in c(1:3))
{
  lwidth<-1
  if (listelement==2)lwidth<-3
  plot(rowSums(out_multi_rothC[[listelement]]$N),axes=FALSE, col=1,type="l", lwd=lwidth,
      lty=listelement+2,cex.lab=2,xlab="",ylab="",ylim=c(0,18))
  par(new=TRUE)
  plot(rowSums(RothC_Cin_ex_sl[[listelement]])/rowSums(RothC_Nin_ex_sl[[listelement]]),
      type="l", lwd=lwidth, lty=listelement+2,axes=FALSE,col=2, cex.lab=2,xlab="",
      ylab="",ylim=c(0,120))
  par(new=TRUE)
}
axis(side=2, pos = 0,
  labels = (0:6)*3, at=(0:6)*20, hadj=0.7, padj = 0.5, cex.axis=2,las=1,col.axis=1)
axis(side=4, pos = 60,
  labels = (0:6)*20, at=(0:6)*20, hadj=0, padj = 0.5, cex.axis=2, las=1,col.axis=2)
axis(side=1, pos = 0,
  labels =  (0:6)*10 , at=(0:6)*10, hadj=0.5, padj = 0, cex.axis=2)
title(ylab=expression("total N  [t ha"^-1*"]"), line=2, cex.lab=2)
mtext("C input / N input", side=4, line=2, cex=2,col=2)
title(xlab="time", line= 2, cex.lab=2)
legend(x=40,y=100,legend = c("normal","half_input","double_Cin"),lty = c(3,4,5),
 lwd=c(1,3,1))
par(oldpar) #back to old par
```

```
#3 Example of RothC application with fictional input
#and fictional measurement data to calculate C0 and N0

#3.1 Input

#fictional data
data(RothC_Cin_ex_sl, RothC_Nin_ex_sl, RothC_site_ex, RothC_env_in_ex, meas_data_ex)

#3.2. Simulation

out_rothC_C0<-sorcering( model="RothC", site=RothC_site_ex, env_in=RothC_env_in_ex,
  t_sim=60, Cin=RothC_Cin_ex, Nin=RothC_Nin_ex, calcC0=TRUE, calcN=TRUE, calcN0=TRUE,
  tsteps="monthly", meas_data=meas_data_ex)

#3.3 Results

#output structure summary
summary(out_rothC_C0)

#example plot
oldpar <- par(no.readonly = TRUE) #save old par
par(mfrow=c(1,1),mar=c(4,4,1,4))
plot(rowSums(out_rothC_C0$N),axes=FALSE, col=1, cex.lab=2,xlab="",ylab="",ylim=c(0,9),
  type="l",lwd=1)
par(new=TRUE)
plot(rowSums(out_rothC_C0$C),axes=FALSE, col=2, cex.lab=2,xlab="",ylab="",ylim=c(0,90),
  type="l",lwd=1)
par(new=TRUE)
plot(x=meas_data_ex[,1],y=meas_data_ex[,3],axes=FALSE, col=1, cex.lab=2,xlab="",ylab="",
  xlim=c(0,length(rowSums(out_rothC_C0$N))),ylim=c(0,9),pch=4,cex=3)
par(new=TRUE)
plot(x=meas_data_ex[,1],y=meas_data_ex[,2],axes=FALSE, col=2, cex.lab=2,xlab="",ylab="",
  xlim=c(0,length(rowSums(out_rothC_C0$N))),ylim=c(0,90),pch=4,cex=3)
par(new=TRUE)
axis(side=2, pos = 0,
  labels = (0:8)*1, at=(0:8)*10, hadj=1, padj = 0.5, cex.axis=2,las=1,col.axis=1)
axis(side=4, pos = 60,
  labels = (0:8)*10, at=(0:8)*10, hadj=0, padj = 0.5, cex.axis=2, las=1,col.axis=2)
axis(side=1, pos = 0,
  labels =  (0:8)*10 , at=(0:8)*10, hadj=0.5, padj = 0, cex.axis=2)
title(ylab=expression("SON [t ha"^-1*"]"), line=2, cex.lab=2)
mtext(expression("SOC [t ha"^-1*"]"), side=4, line=3, cex=2,col=2)
title(xlab="time", line= 2, cex.lab=2)
legend(x=30,y=30,legend = c("model result","measurement"),lwd=c(1,0))
legend(x=31,y=30,legend = c("",""),pch=4,pt.cex=c(0,3),bty="n")
par(oldpar) #back to old par
```

```
#4 Example of Yasso15 application using multiple sites and
#input values of different wood diameters which take uncertainties into account

#4.1 Input

data(Yasso_Cin_ex_wood_u_sl, Yasso_Nin_ex_wood_u_sl, Yasso_C0_ex_sl, Yasso_N0_ex_sl,
  RothC_env_in_ex) #fictional data

#show last entries of C input for 3rd site, 2nd wood layer, 4th uncertainty layer
tail(Yasso_Cin_ex_wood_u_sl[[3]][[2]][[4]])

#diameter of wood input: 2 classes of 0 cm and 10 cm for each of the 3 sites
wood_diam_ex_sl<-list(c(0,10),c(0,10),c(0,10))

#environmental variables
Yasso_env_in_ex<-RothC_env_in_ex[,1:2]

#4.2 Simulation

out_multi_yasso_wood_unc <- sorcering( model="Yasso15", C0_sl=Yasso_C0_ex_sl,
  env_in_sl=rep(list(Yasso_env_in_ex),3), wood_diam_sl=wood_diam_ex_sl,
  t_sim_sl=list(60,60,60), Cin_wood_sl=Yasso_Cin_ex_wood_u_sl,
  Nin_wood_sl=Yasso_Nin_ex_wood_u_sl, N0_sl=Yasso_N0_ex_sl, calcN=TRUE,
  tsteps="monthly", multisite=TRUE, sitelist=list("a","b","c"))

#4.3 Results

#show the last C results for 3rd site, 4th uncertainty layer
tail(out_multi_yasso_wood_unc[[3]][[4]]$C)
```

```
#5 Example of RothC application using stochastically varying parameters
#and multiple sites

#5.1 fictional data
data(RothC_Cin_ex_sl, RothC_Nin_ex_sl, RothC_C0_ex, RothC_N0_ex,
  RothC_site_ex, RothC_env_in_ex)

#standard deviations [%] used for each of the 7 RothC theta parameters
RothC_theta_unc <- c(0,0,1,1,1,1,2)

#5.2 Simulation

out_sl <- sorcering( model="RothC", site_sl=rep(list(RothC_site_ex),3),
  env_in_sl=rep(list(RothC_env_in_ex),3), t_sim_sl=list(60,60,60),
  Cin_sl=RothC_Cin_ex_sl, Nin_sl=RothC_Nin_ex_sl, C0_sl=rep(list(RothC_C0_ex),3),
  N0_sl=rep(list(RothC_N0_ex),3),calcN=TRUE,theta_n_uncertain=10,
  theta_unc=RothC_theta_unc, multisite=TRUE,
  sitelist=list("normal","half_input","double_Cin"))

#5.3 Means and standard deviation

#60 time steps, 5 pools, 9 output types, 10 theta_n_uncertain, 3 sites
out_sl_arr <- array(unlist(out_sl),c(60,5,9,10,3))
out_sl_arr_N <- out_sl_arr[,,2,,] #only output type 2: N
#mean over all uncerts
out_sl_arr_N_mean <- apply( out_sl_arr_N , c(1,2,4), na.rm=TRUE, FUN=mean )

#standard deviation
out_sl_arr_N_sd<-
array(0, dim=c(dim(out_sl_arr_N)[1],dim(out_sl_arr_N)[2],dim(out_sl_arr_N)[4]))
for (dim3 in c(1:dim(out_sl_arr_N)[4]))
  out_sl_arr_N_sd[,,dim3]<-apply(out_sl_arr_N[,,,dim3],c(1:2),sd)

#5.4 Results

#show the last N means for stand 1
tail(out_sl_arr_N_mean[,,1])

#show the last N standard deviations for stand 1
tail(out_sl_arr_N_sd[,,1])
```

```
#6 Example of how to create input lists for a RothC application using stochastically
#varying inputs and input scenarios

#6.1 Input

#fictional data
data(RothC_Cin_ex_sl, RothC_C0_ex, RothC_site_ex, RothC_env_in_ex)

#create input list of 3 scenarios, 100 uncertainties each
set.seed(17) #to make 'random' results reproducible
f1<-1
for (no in c(1:3)) #loop over 3 input scenarios
{
    #normal, half and double input
    Cin <- switch (no, RothC_Cin_ex, RothC_Cin_ex/2, RothC_Cin_ex*2)
    f2 <- 1
    #create fictional uncertainties
    for (unc in c(1:100)) #loop over 100 uncertainties
    {
        randnum<-max(0,rnorm(1,1,0.5)) #out of normal dist. with 50% sd.
        if (f2==1) Cin_u <- list(Cin*randnum) else
        Cin_u[[length(Cin_u)+1]] <- Cin*randnum
        f2 <- 0
    }
    if (f1==1) Cin_u_sl <- list(Cin_u) else
    Cin_u_sl[[length(Cin_u_sl)+1]] <- Cin_u
    f1 <- 0
}

#show input of scenario 3, uncertainty 51
head(Cin_u_sl[[3]][[51]])

#6.2 Simulation
out_sl <- sorcering( model="RothC", site_sl=rep(list(RothC_site_ex),3),
  env_in_sl=rep(list(RothC_env_in_ex),3), t_sim_sl=list(60,60,60),
  Cin_sl=Cin_u_sl, C0_sl=list(RothC_C0_ex,RothC_C0_ex,RothC_C0_ex), tsteps="monthly",
  multisite=TRUE, sitelist=list("normal","half_input","double_Cin"))

#6.3 Means and standard deviation

#60 time steps, 5 pools, 1000 uncertainties, 3 sites
out_sl_arr <- array(unlist(out_sl),c(60,5,100,3))

#means
out_sl_arr_mean <- apply( out_sl_arr , c(1,2,4), na.rm=TRUE, FUN=mean )

#standard deviation
out_sl_arr_sd<-
  array(0, dim=c(dim(out_sl_arr)[1],dim(out_sl_arr)[2],dim(out_sl_arr)[4]))
for (dim3 in c(1:dim(out_sl_arr)[4]))
  out_sl_arr_sd[,,dim3]<-apply(out_sl_arr[,,,dim3],c(1:2),sd)
```

```
#6.4 Results

#C-pool sums of means for the 3 scenarios
totalC_m1<-rowSums(out_sl_arr_mean[,,1])
totalC_m2<-rowSums(out_sl_arr_mean[,,2])
totalC_m3<-rowSums(out_sl_arr_mean[,,3])

#C-pool sums of standard deviations for the 3 scenarios
totalC_s1<-rowSums(out_sl_arr_sd[,,1])
totalC_s2<-rowSums(out_sl_arr_sd[,,2])
totalC_s3<-rowSums(out_sl_arr_sd[,,3])

#example plot
oldpar <- par(no.readonly = TRUE) #save old par
par(mfrow=c(1,1),mar=c(4,4,1,4))
plot(totalC_m1,axes=FALSE, col=2, cex.lab=2,xlab="",ylab="",ylim=c(0,100),
  type="l",lwd=1)
par(new=TRUE)
plot(totalC_m2,axes=FALSE, col=3, cex.lab=2,xlab="",ylab="",ylim=c(0,100),
  type="l",lwd=1)
par(new=TRUE)
plot(totalC_m3,axes=FALSE, col=4, cex.lab=2,xlab="",ylab="",ylim=c(0,100),
  type="l",lwd=1)
par(new=TRUE)
polygon(c(1:60,60:1),c(totalC_m1+totalC_s1, rev(totalC_m1-totalC_s1)),
  border=NA,col=rgb(1,0,0,0.27),density=40,angle=180,xlab="",ylab="")
par(new=TRUE)
polygon(c(1:60,60:1),c(totalC_m2+totalC_s2, rev(totalC_m2-totalC_s2)),
  border=NA,col=rgb(0,1,0,0.27),density=30,xlab="",ylab="")
par(new=TRUE)
polygon(c(1:60,60:1),c(totalC_m3+totalC_s3, rev(totalC_m3-totalC_s3)),
  border=NA,col=rgb(0,0,1,0.27),density=25,angle=90,xlab="",ylab="")
par(new=TRUE)
axis(side=2, pos = 0,
    labels = (0:10)*1, at=(0:10)*10, hadj=1, padj = 0.5, cex.axis=2,las=1,col.axis=1)
axis(side=1, pos = 0,
    labels =  (0:6)*10 , at=(0:6)*10, hadj=0.5, padj = 0, cex.axis=2)
title(ylab=expression("SOC [t ha"^-1*"]"), line=2, cex.lab=2)
title(xlab="time", line= 2, cex.lab=2)
legend(x=20,y=30,fill=c(0,0,0,4,2,3),density=c(0,0,0,25,40,30),angle=c(0,0,0,90,0,45),
  border=c(0,0,0,1,1,1),legend = c("mean double input scenario",
  "mean regular input scenario", "mean half input scneario",
  "uncertainty range double input scenario", "uncertainty range regular input scenario",
  "uncertainty range half input scenario"))
legend(x=20,y=30,lty=c(1,1,1,0,0,0),seg.len=c(1,1,1,0,0,0), col=c(4,2,3,0,0,0),
  legend = c("","","","","",""),bty="n")
par(oldpar) #back to old par
```

References

Coleman, K., Jenkinson, D.S., 1996. RothC-26.3 - A Model for the turnover of carbon in soil, in: Powlson, D.S., Smith, P., Smith, J.U. (Eds.), Evaluation of Soil Organic Matter Models, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 237–246.

Coleman, K., Jenkinson, D.S., 2014. RothC - A model for the turnover of carbon in soil - Model description and users guide. Rothamsted Research Harpenden Herts AL5 2JQ. `https://rothamsted.ac.uk/rothamsted-carbon-model-rothc`.

Dechow, R., Franko, U., Kätterer, T., Kolbe, H., 2019. Evaluation of the RothC model as a prognostic tool for the prediction of SOC trends in response to management practices on arable land. Geoderma 337, 463 – 478.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Bates, D., Chambers, J., 2021a. Rcpp: Seamless R and C++ Integration. R package version 1.0.6, `https://cran.r-project.org/web/packages/Rcpp/index.html`.

Eddelbuettel, D., Francois, R., Bates, D., Ni, B., 2021b. RcppArmadillo: 'Rcpp' Integration for the 'Armadillo' Templated Linear Algebra Library. R package version 0.10.4.0.0, `https://cran.r-project.org/web/packages/RcppArmadillo/index.html`.

Falloon, P., Smith, P., Coleman, K., Marshall, S., 1998. Estimating the size of inert organic matter pool from total soil organic carbon content for use the Rothamsted Carbon Model. Soil Biology & Biochemistry 30, 1207–1211.

Springob, G., Kirchmann, H., 2010. Ratios of carbon to nitrogen quantify non-texture-stabilized organic carbon in sandy soils. Journal of Plant Nutrition and Soil Science 173, 16–18.

Taghizadeh-Toosi, A., 2015. C-TOOL, A simple tool for simulation of soil carbon turnover. Aarhus University, Department of Agroecology. `https://agro.au.dk/fileadmin/DJF/Agro/Medarbejderportal_AGRO/Sektioner/KLIMA/C-TOOL_Documentation.pdf`.

Taghizadeh-Toosi, A., Christensen, B.T., Hutchings, N.J., Vejlin, J., Kätterer, T., Glendining, M., Olesen, J.E., 2014. C-TOOL: A simple model for simulating whole-profile carbon storage in temperate agricultural soils. Ecological Modelling 292, 11 – 25.

Tuomi, M., Thum, T., Järvinen, H., Fronzek, S., Berg, B., Harmon, M., Trofymow, J., Sevanto, S., Liski, J., 2009. Leaf litter decomposition—Estimates of global variability based on Yasso07 model. Ecological Modelling 220, 3362 – 3371.

Viskari, T., Laine, M., Kulmala, L., Mäkelä, J., Fer, I., Liski, J., 2020. Improving Yasso15 soil carbon model estimates with ensemble adjustment Kalman filter state data assimilation. Geoscientific Model Development 13, 5959–5971.

Viskari, T., Pusa, J., Fer, I., Repo, A., Vira, J., Liski, J., 2022. Calibrating the soil organic carbon model Yasso20 with multiple datasets. Geoscientific Model Development 15, 1735–1752.