SKM-Toolbox Manual

July 10, 2012

Contents

1 Background

This manual describes the MATLAB toolbox for structural kinetic modeling (SKM) published by Girbig *et al* (2012b). The principles of structural kinetic modeling are covered elsewhere (Steuer *et al*, 2006; Steuer, 2011) and will not be repeated here in detail.

SKM enables the investigation of dynamical properties of metabolic systems in steady states. When used within a Monte-Carlo framework, it enables the derivation of this information solely based on the structure of the network (given by the stoichiometric matrix $\mathbf{N}$), the steady state concentrations $\mathbf{S}^0$ and the fluxes $\mathbf{v}^0$.

The presented toolbox offers a framework that facilitates the creation and evaluation of structural kinetic models (SK-models). The minimal input necessary to start the program consists of $\mathbf{N}$, $\mathbf{S}^0$ and $\mathbf{v}^0$. Model parameters and their sampling intervals can be derived automatically from the information contained in $\mathbf{N}$. Further settings can be specified by additional input arguments. For example, the manual modification of sampling intervals or the specification of regulatory interactions is performed via an optional input argument.

Details about the SK-model building process, as well as the input and output arguments of the program are explained in section 5.

2 Example scripts

We illustrate the model building and analysis process on three example scripts. These scripts demonstrate how to implement published SK-models using the toolbox:

1. Simplified glycolysis model of Steuer *et al* (2006)
2. Simplified Calvin-Benson cycle model of Steuer *et al* (2006)
3. Detailed Calvin-Benson cycle model of Girbig *et al* (2012a)

3 Function overview

Like Schmidt and Jirstrand (2006), we divide our toolbox functions into three categories: analysis functions, information and editing functions and auxiliary functions.

1. **Analysis functions** are the main functions for creating and analyzing SK-models via Monte-Carlo simulation.
2. **Information and editing functions** are functions that help generating input arguments for the analysis functions and to retrieve and process results.
3. **Auxiliary functions** are support functions for internal use within the toolbox only. They are indicated by the preamble `subFct`. In general, the user is not encouraged to run or modify any auxiliary function manually.

Functions in the first two categories can be invoked directly by the user. Table 1 provides a brief overview of these functions. A detailed description of all functions (except of the auxiliary functions) is given in Appendix A.

4 Installation

The Toolbox requires that MATLAB is already installed on your computer. Basic MATLAB skills are required in order to be able to correctly use the program.

1. Download the `.zip` - file and unpack in your folder of choice (for example, C:\SKM-Toolbox)
2. Start MATLAB
   (a) In order to run the toolbox, make sure that your local path in which you are operating corresponds to the folder containing the skm-function.
   (b) Alternatively, you can also set a a global MATLAB path to this folder (`File -> Set Path...`). We recommend this option because it enables you to access the toolbox from arbitrary locations on your disk.

3. Open the script `'ExampleScript_Steuer2006_Glycolysis.m'`. It demonstrates the process of model building and evaluation on a simple model of the glycolysis pathway taken from the literature (Steuer *et al*, 2006). Go through this script step by step. Further explanations about the individual steps can be found in the Toolbox-Manual (PDF included the downloaded `.zip` - file).

4. After getting acquainted with the basic concepts, try the more complex models provided in example the following example scripts:

   - `'ExampleScript_Steuer2006_CalvinCycle'`: moderately complex literature model (Steuer *et al*, 2006),
   - `'ExampleScript_Girbig2012_CalvinCycle'`: highly complex literature model (Girbig *et al*, 2012a), provided in a separate subfolder within the `.zip` - file together with some assisting functions.

Function Name	Description
	1. Analysis functions
`skm`	Main function of the SKM-toolbox. Creates SK-models, performs Monte-Carlo simulations and plots the distributions of the resulting eigenvalues and model parameters.
	2. Information and editing functions
`skm_assignRandomRegulators`	Assistant function to modify the struct defining sampling intervals for the model parameters by including allosteric regulation at random positions. This struct then serves as an input argument for the `skm`-function.
`skm_countStableModels`	Quantitative evaluation of the number of stable and unstable models. Computes absolute numbers and percentages. Returns estimates and standard deviations.
`skm_createClassifierInput`	Create training data for the `C4.5` classifier for decision trees. Optionally, input for the proprietory `C5.0` classifier can be created instead of `C4.5`.
`skm_createOptionsStruct`	Creates a template for a struct containing the settings for the `skm`-routine. This struct then serves as an input argument for the `skm`-function.
`skm_createParamStruct`	Creates a template for a struct defining the sampling intervals for the model parameters. This struct then serves as an input argument for the `skm`-function.
`skm_pairwiseTests`	Pairwise comparison between the model parameters responsible for stable and unstable models. Uses the Kolmogorov-Smirnov test.
`skm_plotParams`	Plotting the distributions of model parameters and resulting eigenvalues for stable and unstable models.
`skm_readSBML`	Reads an SBML file and extracts the information about metabolite and reactions names, as well as the corresponding stoichiometric matrix for use in the toolbox.

Table 1: Matlab functions for creating and evaluating SK-models by the SKM-toolbox.

5 Getting started

The `skm` function is the main toolbox element for SK-model building and analysis. This section describes its usage, specifically the creation of its input arguments and the handling of its output for further analysis.

5.1 Mandatory input arguments for the `skm`-function

Three input arguments are required for the `skm` function: the **stoichiometric matrix N**, the **steady state concentrations $\mathbf{S^0}$**, and the **steady states fluxes $\mathbf{v^0}$**. For example, the command

```
skm(N, S0, v0)
```

produces 10^4 SK-models in which the numbers and positions of the model parameters are derived automatically from the stoichiometric matrix (negative entries = substrates, positive entries = products). In this (default) case, the model parameters are assumed to represent substrate effects of irreversible enzymatic reactions. Consequently, they are sampled independently from the interval $[0, 1]$ (Steuer *et al*, 2006).

An overview of the three mandatory program input arguments for the `skm`-function is shown in Table 2. All three input arguments `N`, `S0`, `v0` must be MATLAB matrices. `S0` and `v0` each contain a single column or row (columns or rows are both possible because they will be transposed by the program into the right format). The number of rows in `N` equals the length of `S0`, its number of columns equals the length of `v0`.

input argument	explanation	format
`N`	Stoichiometric matrix	Matrix with m rows and r columns
`S0`	Steady state concentrations	Row or column vector of length m
`v0`	Steady state fluxes	Row or column vector of length r

Table 2: Mandatory input arguments for the function `skm`

5.2 Example

The SK-model of a simplified glycolysis pathway in Steuer *et al* (2006) is represented by the following MATLAB matrices:

$$\underset{\texttt{N}}{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ -2 & 0 & 0 & 2 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & -2 & 0 & 0 & 0 & 1 \end{bmatrix}} \quad \underset{\texttt{S0}}{\begin{bmatrix} 5.1 \\ 0.12 \\ 0.0001 \\ 1.48 \\ 2.1 \\ 0.33 \\ 0.67 \\ 1.9 \end{bmatrix}} \quad \underset{\texttt{v0}}{\begin{bmatrix} 50 \\ 50 \\ 80 \\ 80 \\ 60 \\ 20 \\ 20 \\ 60 \end{bmatrix}} \tag{1}$$

The function call

```
eigenvalues = skm(N, S0, v0);
```

starts a Monte-Carlo simulation sampling 12 substrate parameters (one parameter per negative entry in `N`) and reports the results in a message similar to the following:

```
Automatically detecting dependencies between metabolites.
Maximum concentration change in the steady state: max(abs(dS/dt)) = 0.00e+000
Number of sampled model parameters: 12
Iterations: 10000
Results:
Percentage of stable models (max_eig < 2.22e-016): 53.24 +- 2.07 %
Percentage of unstable models (max_eig > 2.22e-016): 46.76 +- 1.20 %
Percentage of unclear models (|max_eig| <= 2.22e-016): 0.00 +- 0.00 %
Elapsed time:  2.01 seconds
```

Line 1 says that during the preprocessing stage, linear dependencies between the rows of the stoichiometric matrix were detected automatically (see Section 6.1 for instructions how to define dependencies manually). In the given network for example, such dependencies exist between the rows 5 and 8 (ATP/ADP), as well as 6 and 7 (NADH/NAD).

Line 2 shows the maximum absolute concentration chances actually occurring for the given stoichiometry and data. Concentration changes are computed by the formula $\frac{d\mathbf{S}}{dt}|_{\mathbf{S}=\mathbf{S}^0} = \mathbf{N} \cdot \mathbf{v}^0$.

Lines 3 and 4 indicate that 10^4 runs of the Monte-Carlo simulation have been performed, sampling 12 model parameters in each run.

The results show that using only substrate parameters representing irreversible kinetics leads to stable models in about 53% of all cases. In general, an SK-model is stable if the largest real part of the eigenvalues is negative. To avoid numerical inaccuracies, we use the MATLAB precision measure `eps` as a threshold to detect values larger or smaller than zero. The standard deviation is computed by partitioning the set of 10^4 SK-models into ten equally sized groups which are evaluated separately. The elapsed time since start of the simulation is shown as well.

In the given example, the `skm` -function returns only one output argument, namely the eigenvalues for each of the 10^4 models. To obtain further information, we can specify several other return values. The complete list of possible output values is explained in the following section.

5.3 Output arguments of the `skm`-function

The `skm`-function returns up to six output arguments. The full function call which returns all output arguments looks as follows:

```
[eigenvalues, stability, modelParam_values, modelParam_names, used_options, used_paramIntervals]
= skm(N, S0, v0)
```

Output argument 1: `eigenvalues` A MATLAB matrix containing the eigenvalues of the Jacobian matrix for each SK-model. The eigenvalues are stored row-wise.

Output argument 2: `stability` A vector indicating the stability of each SK-model. An SK-model is stable if the largest real part of the eigenvalues is negative. Stable models are indicated by entry `1`, unstable models by `0`. Models for which the stability could not be derived in this manner (i.e. the maximum real part equals zero) are indicated by the entry `nan`.

Output argument 3: `modelParam_values` A struct containing the model parameter values which have been randomly sampled in each Monte-Carlo iteration. The struct contains fields for substrate parameter values (`Enzyme_Substrates`), product parameter values (`Enzyme_Products`), as well as parameter values describing regulatory effects (`Regulators`) or further influences (`FurtherParams`).

Output argument 4: `modelParam_names` A struct containing the names of all model parameters which have been randomly sampled in each Monte-Carlo iteration. The struct contains fields for the names of substrate parameters (`Enzyme_Substrates`) sampled from interval $[0, 1]$, product parameters (`Enzyme_Products`) sampled from interval $[-1, 0]$, as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`), both sampled from user-defined intervals. Each field consists of a cell array which denotes the reaction and the metabolite associated with each parameter (columns 1 and 2), as well as the type of interaction the parameter represents in the model (column 3).

Output argument 5: `used_options` A struct containing the settings which have been applied by the algorithm. If the user already specified settings for the `skm`-function by the optional input argument `options` (see section 6.1), `used_options` will be mostly identical to this input argument. However,

settings which were not specified beforehand and which got default values assigned instead are also shown.

Output argument 6: `used_paramIntervals` A struct containing the positions of all model parameters which have been randomly sampled in each Monte-Carlo iteration. The struct contains fields for substrate parameters (`Enzyme_Substrates`) sampled from interval $[0, 1]$, product parameters (`Enzyme_Products`) sampled from interval $[-1, 0]$, as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`), both sampled from user-defined intervals. If the user manually specified parameter positions for the `skm`-function by the optional input argument `paramIntervals` (see section 6.1), `used_paramIntervals` will be mostly identical to this input argument.

6 Specifying toolbox options and manipulating model parameters

6.1 Optional input arguments for the skm-function

Additional optional input arguments allow more detailed specification about how to build and evaluate the SK-models. For example, the full program input in which all arguments are specified looks as follows:

```
skm(N, S0, v0, numModels, options, paramIntervals)
```

Table 3 provides an overview of the optional input arguments and their default values. More detailed descriptions are given in the text below.

input argument	explanation	format	default value
`numModels`	Number of SK-models for Monte-Carlo sampling	integer >= 1	10^4
`options`	Optional parameters	struct	an `options`-struct with default values in each field can be created by the function `skm_createOptionsStruct`
`paramIntervals`	Information about model parameters and their sampling intervals	struct	a `paramIntervals`-struct with default settings in each field can be created by the function `skm_createParamStruct`

Table 3: Optional input arguments for the function `skm`

Optional input 1: `numModels` The number of SK-models created during Monte-Carlo sampling. For each model, the parameters are sampled independently from predefined intervals. Default value: 10^4.

Optional input 2: `options` A struct with optional parameters for model building and analysis. Important notes:

- An `options`-struct with default values in each field can be created by the function `skm_createOptionsStruct`.
- Missing fields will be filled with default values by the `skm`-function.
- If no `options`-argument is specified, the default settings are used.
- If the fields `m_names`, `r_names` or `m_dependent` stay empty (the default case), generic names are assigned to reactions and metabolites, and dependent metabolites are searched automatically by the `skm`-function.

The different fields of the `options`-struct are summarized in Table 4.

Optional input 3: `paramIntervals` A struct which contains separate matrices indicating the positions of different types of model parameters (for example, enzyme substrates, enzyme products, regulators and substrates for mass action kinetics).

The assignment of model parameters and their sampling intervals is the most time-intensive and error-prone process when constructing SK-models. We facilitate this process by dividing the model parameters into different categories. Some of these categories use predefined intervals like $[0, 1]$ for enzyme substrates (derived from the irreversible Michaelis-Menten kinetics, see Steuer *et al* (2006)) whereas other categories apply user-defined intervals.

Table 5 summarizes the different parameter categories and explains their meaning. For each category, the `paramIntervals`-struct contains a matrix of size $r \times m$, where r is the number reactions and m is the number of metabolites. This corresponds to the size of the parameter matrix $\mathbf{T}$ of an SK-model (Steuer *et al*, 2006). Matrix rows represent the network's reactions and columns represent its metabolites.

A model parameter can be included into the model by inserting a corresponding entry into the matrix that best describes the parameter's role in the model. The position of this entry in the matrix depends on the enzyme-metabolite interaction it refers to.

The following section provides detailed instructions about how to create a `paramIntervals`-struct using the function `skm_createParamStruct`.

fieldname	description	possible values	default value
	Options for model structure		
`m_names`	Metabolite names	cell array of strings, same length as `S0`	empty
`r_names`	Reaction names	cell array of strings, same length as `v0`	empty
`m_dependent`	Metabolites with concentrations linearly dependent on other metabolites	boolean matrix consisting of one column or row, same length as `S0`	empty
`m_exclude`	Metabolites to be ignored in the SK-model. Do not combine with field `m_include`!	cell array of strings (names as in the `m_names` argument), OR boolean matrix of same length as `S0`, OR vector of indices.	empty
`m_include`	Metabolites to be explicitly included in the SK-model. If specified, all other metabolites will be ignored. Do not combine with field `m_exclude`!	same as `m_exclude`	empty
`r_exclude`	Reactions to be ignored in the SK-model. Do not combine with field `r_include`!	cell array of strings (names as in the `r_names` argument), OR boolean matrix of same length as `v0`, OR vector of indices.	empty
`r_include`	Reactions to be explicitly included in the SK-model. If specified, all other reactions will be ignored. Do not combine with field `r_exclude`!	same as `r_exclude`	empty
`rm_zero_met`	Ignore metabolites with concentration = 0	`true` or `false`	`false`
`rm_zero_rct`	Ignore reactions with flux = 0	`true` or `false`	`false`
`rm_zero_N`	Ignore metabolites and reactions for which the stoichiometric matrix `N` has only entries = 0	`true` or `false`	`false`
	Options for model parameters		
`kinetics`	Type of basic kinetics assumed for each reaction. Used for the automatic assignment of model parameters to those entries in the struct `paramIntervals`, which have not already been specified by the user (see section 6.2 for details).	`'enzymatic_irrev'`, `'enzymatic_rev'`, `'massAction'`	`'enzymatic_irrev'`
`fct_modify_params`	Handle to a function that performs user-defined post-processing on the parameter matrix before using it for computing the Jacobian matrix (see example script `ExampleScript_Girbig2012_CalvinCycle`).	handle to a function with syntax `f(T, model_indices, options)`	handle to empty function
	Options for program output		
`verbose`	Report status messages in the command window	`true` or `false`	`true`
`plot`	Plot results (see section 7.2 for details).	`true` or `false`	`true`
`balance_stability`	Repeat sampling until equal numbers of stable and unstable models are available. Useful for the creation of balanced training data for machine learning (see section 7.4 for details).	`true` or `false`	`false`
`return_all_eigvals`	Return all eigenvalues of each Jacobian matrix. If `false`, only the eigenvalues with largest real part (which is sufficient to determine stability) are computed in each iteration, using the command `eigs(J, 1, 'lr')`. For large matrices, this requires less runtime (see section 7.5 for details). If `true`, the full set of eigenvalues are computed using `eig(J)`.	`true` or `false`	`true`

Table 4: Fields in the `options`-struct (optional input argument for the `skm`-function).

fieldname	description	Sampling interval
`Enzyme_Substrates`	Boolean matrix indicating positions of enzyme substrates. In reversible reactions (indicated by `options.kinetics = 'enzymatic_rev'`), substrate parameters will have antagonistic effects on the forward- and backward reaction.	$[0, 1]$
`Enzyme_Products`	Boolean matrix indicating positions of enzyme products. In reversible reactions, product parameters will have antagonistic effects on the forward- and backward reaction.	$[-1, 0]$
`Constant_Params`	Numerical matrix indicating positions and values of constant parameters (e.g. parameters describing mass-action kinetics have a fixed value of 1)	no sampling intervals, parameters fixed to user-defined values
`Regulators`	Numerical matrix indicating positions and interval boundaries of regulators (define negative boundaries for inhibitors and positive boundaries for activators) In reversible reactions, regulatory parameters influence forward- and backward reaction in the same manner.	user-defined
`FurtherParams_lower`, `FurtherParams_upper`	Numerical matrices indicating upper and lower interval boundaries of parameters describing arbitrary effects. In case of reversible reactions, parameters will only be affect the direction for which they are specified and not included into the opposite direction. Can be used for example to model transport reactions involving different compartments.	user-defined

Table 5: Fields in the `paramIntervals`-struct (optional input argument for the `skm`-function).

6.2 Automatic creation of a `paramIntervals`-struct

The automatic detection of model parameters and their sampling intervals is the most complex part of the algorithm. Since the assignment of model parameters by hand is a time-intensive and error-prone process, we offer an automated procedure that derives parameter positions directly from the network structure.

This is achieved by the function `skm_createParamStruct`, which derives the positions of substrates and products from the stoichiometric matrix `N`, translates them into model parameters and returns a struct which can be used as a `paramIntervals`-argument for the `skm`-function. This resulting `paramIntervals`-struct can then be further modified by the user, for example by adding regulatory interaction or parameters which are fixed to constant values.

The following code demonstrates how to create a `paramIntervals`-struct using this function:

```
paramIntervals = skm_createParamStruct(N, kinetics)
```

For the automatic assignment of sampling intervals by the function `skm_createParamStruct`, we need to specify the type of reactions involved in the network. This is given by the input argument `kinetics`. If this argument is left out, irreversible enzymatic reactions will be assumed by default. Currently, three types of reactions can be specified:

1. **Irreversible enzyme reactions** (`kinetics='enzymatic_irrev'`): each substrate (indicated by negative entries in `N`) leads to a parameter sampled from interval $[0, 1]$. Products are ignored. If used in `skm_createParamStruct`, the field `Enzyme_Substrates` of the resulting `paramIntervals`-object contains a 1 for each substrate. All other matrices contain only 0s.

2. **Mass action kinetics** (`kinetics='massAction'`): each substrate (indicated by negative entries in `N`) leads to fixed parameter of 1. Products are ignored. If used in `skm_createParamStruct`, the field `Constant_Params` of the resulting `paramIntervals`-object contains a 1 for each substrate. All other matrices contain only 0s.

3. **Reversible enzyme reactions** (`kinetics='enzymatic_rev'`): This mode requires the reactions to be split into a forward and backward part. Each substrate (indicated by negative entries for the forward reactions in `N`) leads to a parameter sampled from $[0, 1]$. Each product (indicated by

positive entries for the forward reactions in `N`) leads to a parameter sampled from $[-1, 0]$. If used in `skm_createParamStruct`, the field `Enzyme_Substrates` of the resulting `paramIntervals`-object contains a 1 for each substrate.The field `Enzyme_Products` contains a 1 for each product. All other matrices contain only 0s. Since each substrate-parameter θ_S describes positive influence on the forward reaction rate, it also implies a negative influence on the backwards reaction rate. The `skm`-routine will therefore assign the parameter to two positions in the parameter matrix $\mathbf{T}$: the positive value θ_S for the forward reaction, and the negative value $\theta_S - 1$ for the backward reaction. Product parameters are handled in a similar manner (see Grimbs *et al* (2007) for details).

6.3 When do I need to specify a `paramIntervals`-input argument?

If no modifications of the struct are planned by the user and the automatically derived parameters are sufficient, there is no need to generate a `paramIntervals`-struct before starting the analysis. If not provided as an optional input argument, the `skm`-function will automatically create such an object internally. In doing so, it will use the reaction type which is specified in `options.kinetics` and apply the above mentioned rules.

6.4 What if I only want to specify some parameters while the rest should be derived automatically?

Often it is sufficient to just modify some specific parameters while the rest can be derived automatically based on the entries in `N`. Examples comprise the insertion of regulatory interactions into the `Regulators` field or of a constant parameter into the field `Constant_Params`. There are two ways to handle this:

1. Create a `paramIntervals`-struct using the function `skm_createParamStruct`. Modify the necessary matrices and submit to the function `skm`.
2. Manually create a `paramIntervals`-struct that contains only those matrices necessary to describe the desired parameters. For example, for regulatory parameters it is enough to specify the field `paramIntervals.Regulators`. All other fields will be automatically created and filled with model parameters during runtime.

6.5 Example

The stoichiometric matrix of the simplified glycolysis model of Steuer *et al* (2006) has already been described in Section 5.2. Assuming irreversible enzyme-catalyzed reactions, the command

```
param_intervals = skm_createParamStruct(N, 'enzymatic_irrev')
```

returns a struct in which the positions of the enzymes' substrates are specified in the field `Enzyme_Substrates`:

`param_intervals.Enzyme_Substrates` =

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

Starting the `skm` function with this `struct` as an input argument results in $\sim 53\%$ stable models (see Section 5.2). We can investigate the effects of additional interactions that take place in the network by manually manipulating the remaining fields in the `struct`.

So far, all other fields except `Enzyme_Substrates` are still empty. However, we can include product feedback effects in the field `Enzyme_Products`. We want to assign a product parameter for each positive entry in $\mathbf{N}$:

```
param_intervals.Enzyme_Products = N'>0
```

This command produces a matrix which indicates the positions of all parameters describing product inhibition:

param_intervals.Enzyme_Products =

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

These parameters will be sampled from the interval $[-1, 0]$ during the Monte-Carlo simulation. Using these settings, the number of stable models produced in the Monte-Carlo simulation rises to $\sim 95\%$!

Furthermore, we can add an additional regulatory parameter for feedback inhibition of FBPase by ATP:

```
param_intervals.Regulators(1, 5) = -2
```

The `Regulators` field now contains the lower boundary of the sampling intervals (-2) for this parameter:

param_intervals.Regulators =

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

This regulatory parameter decreases the number of stable models to $\sim 90\%$.

6.6 Including allosteric regulators at random positions

An already existing `paramIntervals`-struct can be extended to include regulatory parameters at random network positions. This is done by the function `skm_assignRandomRegulators` . It returns a matrix that can be assigned to the `Regulators`-field of the struct. If some regulatory interactions have already been assigned manually, the newly created matrix can be added to the existing entry in the `Regulators`-field.

7 Working with the results

7.1 Plotting the results

In order to illustrate the results of the Monte-Carlo experiments graphically, the toolbox function `skm_plotParams` creates plots which show the distributions of the model parameters and of the resulting eigenvalues for stable and unstable models. When `options.plot = TRUE` (the default), the `skm`-function invokes this function automatically after finishing Monte-Carlo sampling. A detailed documentation of the function is provided in Appendix Section A.2.7.

7.2 Pairwise comparison of the parameters

The function `skm_pairwiseTests` compares the model parameters responsible for stable and unstable models by the Kolmogorov-Smirnov test. The syntax looks as follows:

```
[p_values_sorted, names_sorted] = skm_pairwiseTests(modelParam_values, stability, modelParam_names)
```

A detailed documentation of this function is provided in Appendix Section A.2.6.

7.3 Example

As described in section 5.2, analysing the glycolysis model of Steuer *et al* (2006) using default settings for the `skm`-function results in $\sim$ 53% stable models. The corresponding eigenvalue and parameter distributions are shown in Figure 1.

Pairwise comparison of the model parameters yields the following results:

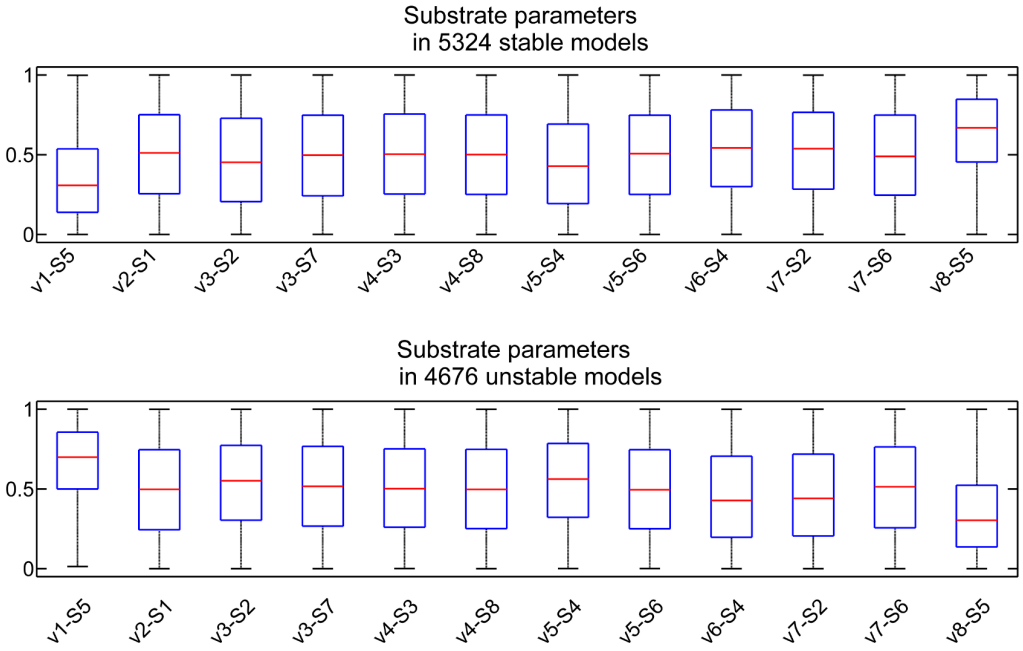
$$\begin{array}{cc} \texttt{p\_values\_sorted} & \texttt{names\_sorted} \\ \\ \begin{bmatrix} 0 \\ 0 \\ 4.93 \cdot 10^{-44} \\ 1.62 \cdot 10^{-43} \\ 9.50 \cdot 10^{-32} \\ 2.79 \cdot 10^{-13} \\ 0.0275 \\ 0.3066 \\ 0.4726 \\ 0.6335 \\ 0.6826 \\ 0.9321 \end{bmatrix} & \begin{matrix} \{\text{'v1 - S5 - Substrate'} \\ \text{'v8 - S5 - Substrate'} \\ \text{v6 - S4 - Substrate'} \\ \text{v5 - S4 - Substrate'} \\ \text{v3 - S2 - Substrate'} \\ \text{v7 - S2 - Substrate'} \\ \text{v3 - S7 - Substrate'} \\ \text{v7 - S6 - Substrate'} \\ \text{v2 - S1 - Substrate'} \\ \text{v5 - S6 - Substrate'} \\ \text{v4 - S3 - Substrate'} \\ \text{v4 - S8 - Substrate'}\} \end{matrix} \end{array}$$

Thus, the first six parameters exhibit significant influence on stability (significance threshold $\alpha = 0.01$). The parameters associated with reaction v1 and metabolite S5 (FBPase-ATP), as well as with v8 and S5 (ATPase-ATP) show the smallest p-values. The strong impact of these two parameters on stability can also be observed in Figure 1a), where they show the most striking differences between stable and unstable models.

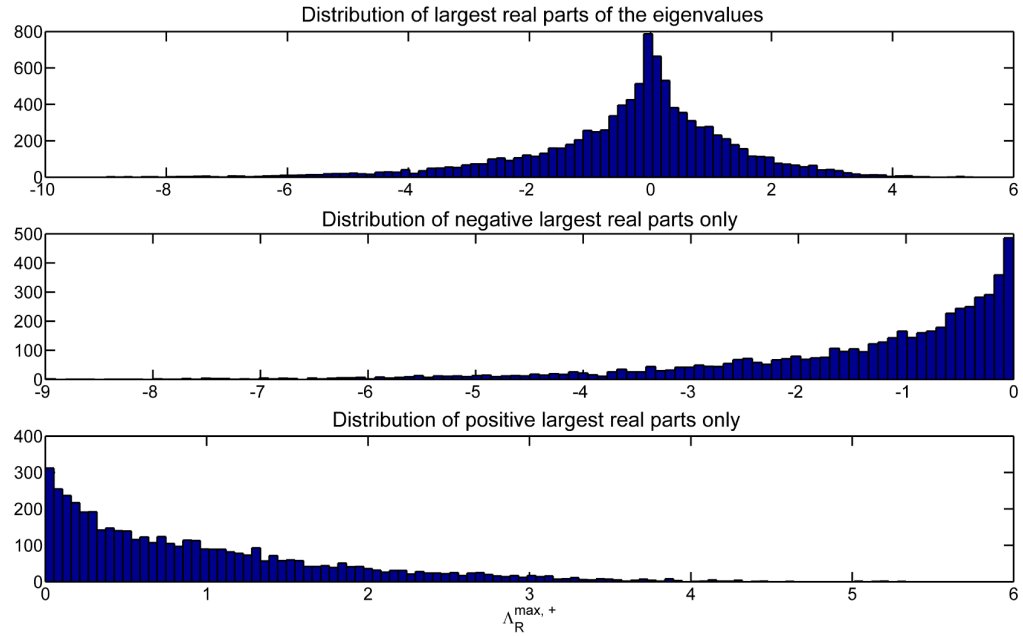
7.4 Creating training data for the `C4.5` or `C5.0` algorithm for decision trees

The information about which model parameters lead to stable and unstable models can be used for classifier training. In doing so, the function `skm_createClassifierInput` enables the creation of training data files for the decision tree algorithms C4.5 (Quinlan, 1993). Optionally, input for the proprietory `C5.0` algorithm (Quinlan, 2012) can be created instead. A detailed documentation of the function is provided in Appendix Section A.2.3.

To avoid bias, we recommend to use a balanced dataset with equal numbers or stable and unstable models for classifier training. This can be obtained by setting `options.balance_stability = TRUE` before starting the `skm`-function.



(a)



(b)

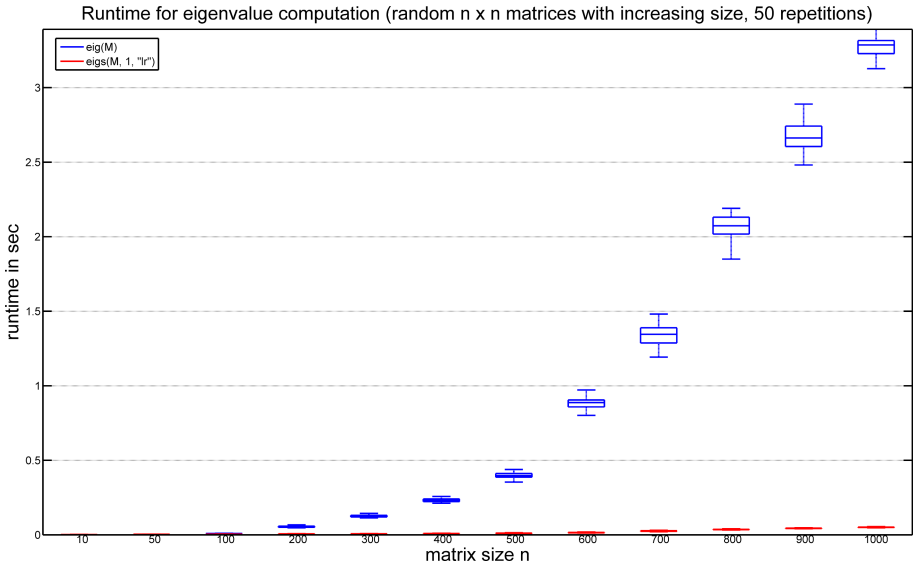
Figure 1: Graphical output produced by toolbox function `skm_plotParams`. (a): Boxplots of the model parameters resulting in stable and unstable models (b): Histograms of the corresponding eigenvalue distributions.

7.5 Obtaining further information from the eigenvalues

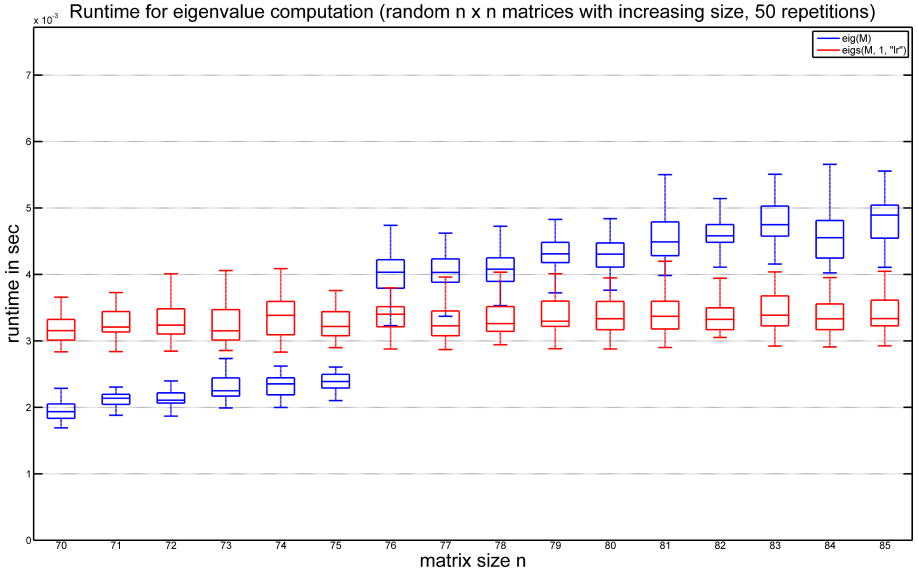
When evaluating the Jacobian matrix of a metabolic system in a steady state, the eigenvalue with largest real part indicates whether the steady state is stable. The remaining eigenvalues, however, can provide additional information. For example, the script `ExampleScript_Steuer2006_CalvinCycle` demonstrates how the two eigenvalues with largest real parts can be used to detect Hopf bifurcations.

By default, the toolbox applies the `eig` function for eigenvalue computation. However, for large systems, computation of the full set of eigenvalues using the `eig` function can be very runtime-intensive. We therefore offer the option `options.return_all_eigvals = false`, which activates usage of the `eigs` function for obtaining only the eigenvalue with maximum real part. We recommend using this option whenever (1) the system is large, and (2) stability is the only target variable of the analysis.

In order to obtain a quantitative threshold for when to chose this setting, we performed a simulation study, the results of which are shown in Figure 2a). It revealed that in contrast to the `eig` function, which is clearly dependent on the matrix size, the `eigs` function keeps relatively constant runtime for increasing matrix sizes. A closer look (Fig. 2b) reveals that this leads to a better runtime in systems with 76 metabolites ore more.



(a)



(b)

Figure 2

A Function documentation in detail

This section provides detailed documentation of all toolbox functions which can be directly invoked by the user. A graphical overview of the toolbox pipeline and the roles of the different functions is given in Figure 3.

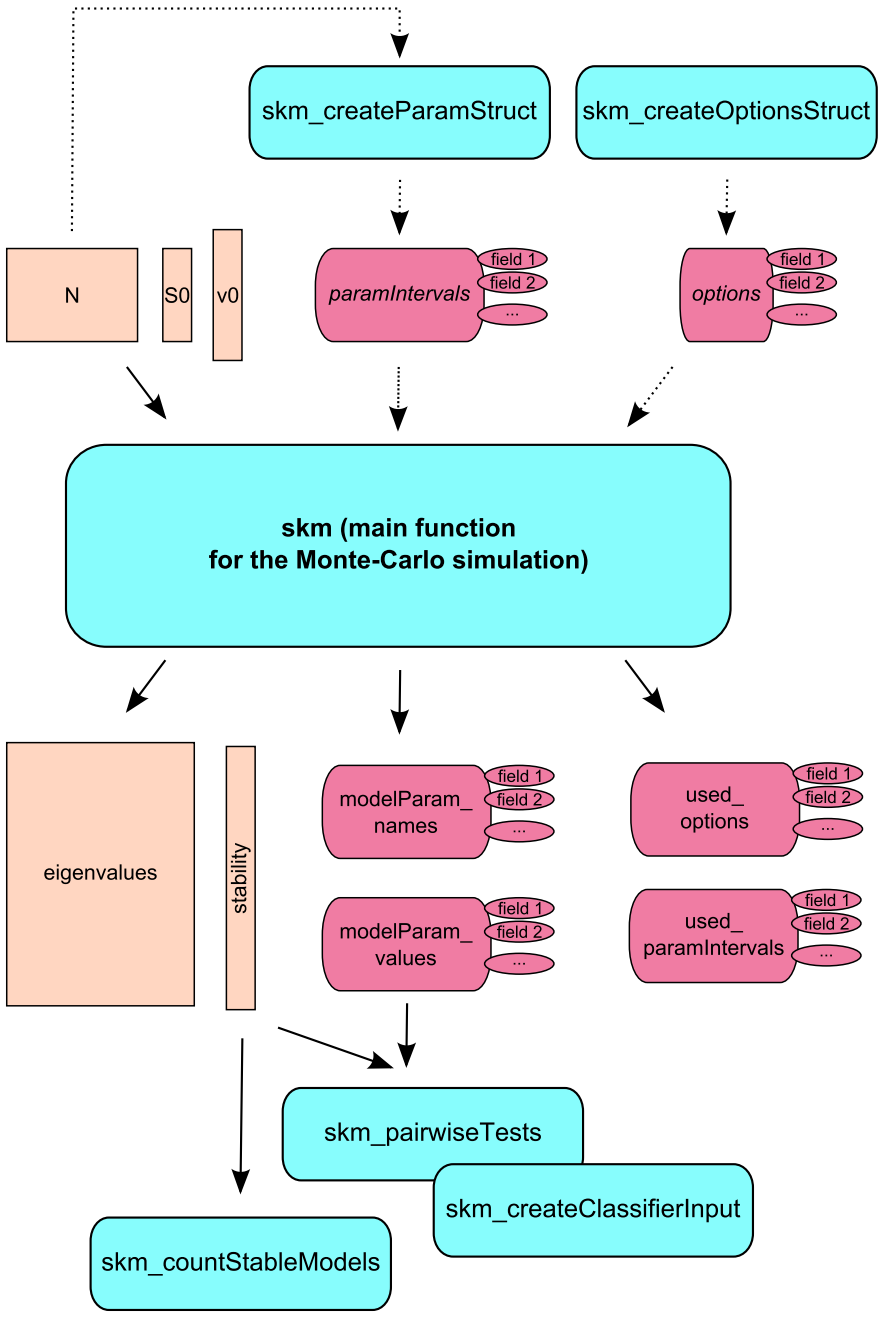


Figure 3: Scheme of the overall structure of the toolbox and the roles of its individual functions. Toolbox functions are shown in blue; input and output arguments are shown in orange (MATLAB matrices) or red (MATLAB structs). Optional input arguments for a function a written in *italics* and indicated by dotted arrows.

In the following documentation, mandatory input arguments are marked by *.

A.1 Analysis functions

Analysis functions are the main functions for creating and analyzing SK-models via Monte-Carlo simulation.

A.1.1 skm

The main function of the SKM-toolbox. Creates SK-models, performs Monte-Carlo simulations and plots the resulting distributions of eigenvalues and model parameters. Detailed explanations of its use together with an example are provided in Section 5.

```
[eigenvalues, stability, modelParam_values, modelParam_names, used_options, used_paramIntervals] = ...
skm(N, S0, v0, numModels, options, paramIntervals)
```

Function `subFct_reduce_T`	
	Input arguments:
`N`*	Stoichiometric matrix with m rows and r columns. Format: matrix.
`S0`*	Row or column vector of m steady state concentrations.
`v0`*	Row or column vector of r steady state fluxes.
`numModels`	Number of SK-models created during Monte-Carlo sampling (optional argument, default value: 10^4). Format: integer.
`options`	Settings for the `skm`-routine. See Section 6.1 for details (optional argument, default values are listed in Table 4). Format: struct.
`paramIntervals`	A `struct` with information about the model parameters and their sampling intervals. See Section 6.1 for details. It gives the positions of substrate parameters (`Enzyme_Substrates`) sampled from interval $[0, 1]$, product parameters (`Enzyme_Products`) sampled from interval $[-1, 0]$, as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`), both sampled from user-defined intervals. If not specified, default settings are assigned internally using the function `skm_createParamStruct` for the kinetic rate law specified in `options.kinetics`.
	Output arguments:
`eigenvalues`	A matrix containing the eigenvalues (stored row-wise) of the Jacobian of each SK-model created during Monte-Carlo simulation.
`stability`	Indicator of stability of each SK-model. An SK-model is stable if the largest real part of the eigenvalues is negative. Stability = `1`, instability = `0`. Models for the maximum real part equals zero are indicated by `nan`. Format: column vector.
`modelParam_values`	A `struct` of model parameter values which have been randomly sampled in each Monte-Carlo iteration. Contains the values of substrate parameters (`Enzyme_Substrates`), product parameters (`Enzyme_Products`), as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`).
`modelParam_names`	A `struct` with the names of all model parameters which have been randomly sampled in each Monte-Carlo iteration. Contains the names of substrate parameters (`Enzyme_Substrates`), product parameters (`Enzyme_Products`), as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`). Each field consists of a cell array which denotes the affected reaction (column 1) and metabolite (column 2), as well as the role of the parameter in the model (column 3).
`used_options`	A `struct` with the settings that have actually been applied by the algorithm. Its fields correspond to those of the `options` input argument.
`used_paramIntervals`	A `struct` with the parameter sampling intervals that have actually been used in the algorithm.Its fields correspond to those of the `paramIntervals` input argument.

A.2 Information and editing functions

Information and editing functions are functions that help generating input arguments for the analysis functions and to retrieve and process results.

A.2.1 skm_assignRandomRegulators

Creates a matrix with regulatory parameters at random network positions that can be assigned to the `Regulators`-field of a `paramInterval`-struct (see Section 6.6 for details).

```
[RegulatorMatrix, positions, values] = ...
skm_assignRandomRegulators(networkDimensions, intervalLimits, kinetics)
```

Function `skm_assignRandomRegulators`	
	Input arguments:
`networkDimensions`'	Vector $[m, r]$ where m is the number of metabolites and r is the number of reactions in the network.
`intervalLimits`*	Vector with positive / negative entries specifying the boundaries for activating or inhibiting regulatory parameters.
`kinetics`	Type of enzyme kinetic rate law assumed for each reaction. Possible values: `'enzymatic_irrev'`, `'enzymatic_rev'`, `'massAction'` (see Section 6.2 for details). This is an optional argument. If not specified, default value `'enzymatic_irrev'` is used.
	Output arguments:
`RegulatorMatrix`	Matrix indicating the positions and sampling interval boundaries of regulatory parameters. It can be assigned to the `Regulators`-field of an existing `paramIntervals`-struct.
`positions`	Row and column coordinates of all randomly added regulatory parameters. Format: matrix with two columns (1st column: row indices, 2nd column: column indices).
`values`	Interval boundaries of all randomly added regulatory parameters. Format: vector.

A.2.2 skm_countStableModels

Quantitative evaluation of the number of stable and unstable models. Computes absolute numbers and percentages. Returns estimates and standard deviations.

```
[stable_percent_mean, stable_percent_sd, stable_models_mean, stable_models_sd] = ...
skm_countStableModels(stability)
```

Function `skm_countStableModels`	
	Input arguments:
`stability`*	Indicator of stability of each SK-model. Stable models are indicated by entry `1`, unstable models by `0`. Models for which the stability could not be derived in this manner (i.e. the maximum real part equals zero) are indicated by `nan`. Format: column vector.
	Output arguments:
`stable_percent_mean`	Average percentage proportion of stable models.
`stable_percent_sd`	Standard deviation of the percentage proportion of stable models.
`stable_models_mean`	Average absolute number of stable models.
`stable_models_sd`	Standard deviation of the absolute number of stable models.

A.2.3 skm_createClassifierInput

Create training data for the `C4.5` classifier for decision trees. Optionally, input for the proprietory `C5.0` classifier can be created instead of `C4.5`. This function creates a `.names` and `.data` file. Optionally, a `.test` file with separate test data can be created as well.

```
skm_createClassifierInput(paramValues_train, labels_train, modelParam_names, filename, ...
classifier, classNames, paramValues_test, labels_test)
```

Function `skm_createClassifierInput`	
	Input arguments:
`paramValues_train`*	Model parameter values which should be used as training data. Format: struct with same field names as `skm`-function output `modelParam_values`.
`labels_train`*	Class labels (i.e. indicators of stability) for the training data.
`modelParam_names`*	Names of the model parameters in the training and test data. Format: struct with same field names as `skm`-function output `modelParam_names`.
`filename`*	Path for storing the resulting files on disc.
`classifier`	Decision tree algorithm to be applied. Possible values: `'C4.5'` , `'C5.0'`. Default: `'C4.5'`.
`classNames`	Names of the classes. Format: cell. Default: `{'UNSTABLE','STABLE'}`
`paramValues_test`	Optional model parameter values for the creation of test data. Format: struct with same field names as `skm`-function output `modelParam_values`.
`labels_test`	Class labels (i.e. indicators of stability) for the optional test data.
	Output arguments:
	-

A.2.4 skm_createOptionsStruct

Assistant function creating a template struct that contains all default settings for the toolbox. This struct then serves as an input argument for the `skm` - function. Settings can be adapted by the user by manually modifying the corresponding fields in the struct.

```
options = skm_createOptionsStruct()
```

Function `skm_createOptionsStruct`	
	Input arguments:
	-
	Output arguments:
`options`	Default settings for the `skm`-routine (see Table 4 for a comprehensive list of all entries). Format: struct.

A.2.5 skm_createParamStruct

Assistant function creating a template struct that defines network positions and sampling intervals of the model parameters. This struct then serves as an input argument for the `skm` - function. Parameters can adapted by the user by manually modifying the corresponding fields in the struct.

```
paramIntervals = skm_createParamStruct(N, kinetics)
```

Function `skm_createParamStruct`	
	Input arguments:
`N`*	Stoichiometric matrix with m rows and r columns.
`kinetics`	Type of enzyme kinetic rate law assumed for each reaction. Possible values: `'enzymatic_irrev'`, `'enzymatic_rev'`, `'massAction'` (see Section 6.2 for details). This is an optional argument. If not specified, default value `'enzymatic_irrev'` is used.
	Output arguments:
`paramIntervals`	Suggestions for positions and sampling intervals of the model parameters (see Section 6.1 for details). Format: struct.

A.2.6 skm_pairwiseTests

Pairwise comparison between the model parameters responsible for stable and unstable models. Uses the Kolmogorov-Smirnov test.

```
[p_values_sorted, names_sorted] = skm_pairwiseTests(modelParam_values, stability, modelParam_names)
```

Function `skm_pairwiseTests`	
	Input arguments:
`modelParam_values`*	Model parameter values which have been randomly sampled in each Monte-Carlo iteration. Format: struct. It gives the values of substrate parameters (`Enzyme_Substrates`), product parameters (`Enzyme_Products`), as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`).
`stability`*	Indicator of stability of each SK-model. Stable models are indicated by entry `1`, unstable models by `0`. Models for which the stability could not be derived in this manner (i.e. the maximum real part equals zero) are indicated by `nan`. Format: column vector.
`modelParam_names`*	Names of all model parameters which have been randomly sampled in each Monte-Carlo iteration. Format: struct. It gives the names of substrate parameters (`Enzyme_Substrates`), product parameters (`Enzyme_Products`), as well as parameters describing regulatory effects (`Regulators`) or further influences (`FurtherParams`). Each field consists of a cell array which denotes the reaction (column 1) and metabolite (column 2) associated with each parameter, as well as its role in the model (column 3).
	Output arguments:
`p_values_sorted`	p-values for the individual model parameters sorted in ascending order. Format: vector.
`names_sorted`	The corresponding model parameters sorted in the same order as the p-values. Format: cell array.

A.2.7 skm_plotParams

Plotting the distributions of model parameters and resulting eigenvalues for stable and unstable models.

```
figure_handles = skm_plotParams(modelParam_values, eigenvalues, stability, modelParam_names)
```

Function skm_plotParams	
	Input arguments:
modelParam_values*	Model parameter values which have been randomly sampled in each Monte-Carlo iteration. Format: struct. It gives the values of substrate parameters (Enzyme_Substrates), product parameters (Enzyme_Products), as well as parameters describing regulatory effects (Regulators) or further influences (FurtherParams).
eigenvalues*	Eigenvalues of the Jacobian matrix of each SK-model created during Monte-Carlo simulation. Format: matrix, eigenvalues are stored row-wise.
stability *	Indicator of stability of each SK-model. Stable models are indicated by entry 1, unstable models by 0. Models for which the stability could not be derived in this manner (i.e. the maximum real part equals zero) are indicated by nan. Format: column vector.
modelParam_names*	Names of all model parameters which have been randomly sampled in each Monte-Carlo iteration. Format: struct. It gives the names of substrate parameters (Enzyme_Substrates), product parameters (Enzyme_Products), as well as parameters describing regulatory effects (Regulators) or further influences (FurtherParams). Each field consists of a cell array which denotes the reaction (column 1) and metabolite (column 2) associated with each parameter, as well as its role in the model (column 3).
	Output arguments:
figure_handles	Handles of all figures created by the function. Format: vector.

A.2.8 skm_readSBML

This function enables the extraction of metabolites, reactions, and stoichiometric matrix from an SBML file for use in the SKM toolbox. It uses the TranslateSBML function of the LibSBML package (Bornstein *et al*, 2008). This Package can be downloaded under http://sbml.org/Software/libSBML. When installing LibSBML, make sure that you include its MATLAB API.

```
[m_names, r_names, N] = skm_readSBML(SBML_file)
```

Function skm_readSBML	
	Input arguments:
SBML_file*	Name of SBML file.
	Output arguments:
m_names	Cell array with metabolite names.
r_names	Cell array with reaction names.
N	Stoichiometric matrix.

References

Bornstein BJ, Keating SM, Jouraku A, Hucka M (2008) LibSBML: An API library for SBML. *Bioinformatics* **24**: 880–881

Girbig D, Grimbs S, Selbig J (2012a) Systematic Analysis of Stability Patterns in Plant Primary Metabolism. *PLoS ONE* **7**: e34686

Girbig D, Selbig J, Grimbs S (2012b) A matlab toolbox for structural kinetic modeling. *submitted*

Grimbs S, Selbig J, Bulik S, Holzhütter H, Steuer R (2007) The stability and robustness of metabolic states: identifying stabilizing sites in metabolic networks. *Molecular Systems Biology* **3**: 146

Quinlan JR (1993) *C4.5: programs for machine learning*. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, revised edition

Quinlan JR (2012) *Data Mining Tools See5 and C5.0*. Last accessed 2012 Mar 10

Schmidt H, Jirstrand M (2006) Systems Biology Toolbox for MATLAB: A computational platform for research in systems biology. *Bioinformatics* **22**: 514–515

Steuer R (2011) Exploring the Dynamics of Large-Scale Biochemical Networks: A Computational Perspective. *The Open Bioinformatics Journal* **5**: 4–15

Steuer R, Gross T, Selbig J, Blasius B (2006) Structural kinetic modeling of metabolic networks. *Proceedings of the National Academy of Sciences of the United States of America* **103**: 11868–11873