CTSM-R User Guide

Rune Juhl Niels Rode Kristensen Peder Bacher
Jan Kloppenborg Henrik Madsen

April 3, 2013

Contents

Chapter 1

Introduction

This is the user guide for the R package CTSM-R containing an implementation of the modeling software Continuous Time Stochastic Modelling (CTSM), which has, until now, been available only with a GUI. CTSM-R provides script based modeling through R and thereby enables a much more efficient work flow. The present user guide contains the information needed to use CTSM-R. The mathematics behind CTSM are described in [2]. Examples of using CTSM-R for different modeling applications can be found in separate documents, which can be downloaded at the website[1]. The examples features modeling of the heat dynamics of buildings and are accompanied with R scripts and provide a quick way of getting started. They are listed with short descriptions in Section 3.12. For information and download of R, see the website[2])

This user guide contains: Section 2 with the instructions for installation of the package. Section 3 with a description of the basic usage. Finally, in Section 4 tips and tricks for modeling with R and CTSM-R are given.

1.1 Getting started

In the following a brief mathematical description of the features in CTSM-R is given. A complete description of the mathematics behind the algorithms of the program can be found in the Mathematics Guide [2].

1.2 Features of CTSM-R

CTSM-R provides features for performing *maximum likelihood* (ML) and *maximum a posteriori* (MAP) estimation of unknown parameters of continuous-discrete stochastic state space models. Continuous-discrete stochastic state space models are models that consist of a set of stochastic differential equations describing the dynamics of a system in continuous time and a set of algebraic equations describing how measurements are obtained at discrete time instants. Several independent data sets can be used for the estimation, and the program also provides features for dealing with varying sample times, occa-

[1] `www.ctsm.info`

[2] `www.r-project.org`

sional outliers and missing observations. In addition various statistical tests and residual analysis can be carried out with the program.

Model structures

Within CTSM-R the class of continuous-discrete stochastic state space models is divided into three distinct classes. The class of the model has influence on several aspects of the applied computations, see [2] for details. In CTSM-R the class is automatically detected from the specification of the model (opposed to the CTSM GUI version, where the model is specified differently depending on the class).

The class of nonlinear (NL) models

The class of NL models is the class of models described by the equations:

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, t, \boldsymbol{\theta})dt + \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta})d\boldsymbol{\omega}_t \tag{1.1}$$

$$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) + \boldsymbol{e}_k \tag{1.2}$$

where $t \in \mathbb{R}$ is time, $\boldsymbol{x}_t \in \mathbb{R}^n$ is a vector of state variables, $\boldsymbol{u}_t \in \mathbb{R}^m$ is a vector of input variables, $\boldsymbol{y}_k \in \mathbb{R}^l$ is a vector of output variables, $\boldsymbol{\theta} \in \mathbb{R}^p$ is a vector of parameters, $\boldsymbol{f}(\cdot) \in \mathbb{R}^n$, $\boldsymbol{\sigma}(\cdot) \in \mathbb{R}^{n \times n}$ and $\boldsymbol{h}(\cdot) \in \mathbb{R}^l$ are nonlinear functions, $\{\boldsymbol{\omega}_t\}$ is an n-dimensional standard Wiener process and $\{\boldsymbol{e}_k\}$ is an l-dimensional white noise process with $\boldsymbol{e}_k \in N(\boldsymbol{0}, \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}))$.

The class of linear time-varying (LTV) models

The class of LTV models is the class of models described by the equations:

$$d\boldsymbol{x}_t = (\boldsymbol{A}(\boldsymbol{x}_t, \boldsymbol{u}_t, t, \boldsymbol{\theta})\boldsymbol{x}_t + \boldsymbol{B}(\boldsymbol{x}_t, \boldsymbol{u}_t, t, \boldsymbol{\theta})\boldsymbol{u}_t)\, dt + \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta})d\boldsymbol{\omega}_t \tag{1.3}$$

$$\boldsymbol{y}_k = \boldsymbol{C}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k, \boldsymbol{\theta})\boldsymbol{x}_k + \boldsymbol{D}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k, \boldsymbol{\theta})\boldsymbol{u}_k + \boldsymbol{e}_k \tag{1.4}$$

where $t \in \mathbb{R}$ is time, $\boldsymbol{x}_t \in \mathbb{R}^n$ is a state vector, $\boldsymbol{u}_t \in \mathbb{R}^m$ is an input vector, $\boldsymbol{y}_k \in \mathbb{R}^l$ is an output vector, $\boldsymbol{\theta} \in \mathbb{R}^p$ is a parameter vector, $\boldsymbol{A}(\cdot) \in \mathbb{R}^{n \times n}$, $\boldsymbol{B}(\cdot) \in \mathbb{R}^{n \times m}$, $\boldsymbol{\sigma}(\cdot) \in \mathbb{R}^{n \times n}$, $\boldsymbol{C}(\cdot) \in \mathbb{R}^{l \times n}$ and $\boldsymbol{D}(\cdot) \in \mathbb{R}^{l \times m}$ are nonlinear functions, $\{\boldsymbol{\omega}_t\}$ is an n-dimensional standard Wiener process and $\{\boldsymbol{e}_k\}$ is an l-dimensional white noise process with $\boldsymbol{e}_k \in N(\boldsymbol{0}, \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}))$.

The class of linear time invariant (LTI) models

The class of LTI models is the class of models described by the equations:

$$d\boldsymbol{x}_t = (\boldsymbol{A}(\boldsymbol{\theta})\boldsymbol{x}_t + \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u}_t)\, dt + \boldsymbol{\sigma}(\boldsymbol{\theta})d\boldsymbol{\omega}_t \tag{1.5}$$

$$\boldsymbol{y}_k = \boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x}_k + \boldsymbol{D}(\boldsymbol{\theta})\boldsymbol{u}_k + \boldsymbol{e}_k \tag{1.6}$$

where $t \in \mathbb{R}$ is time, $\boldsymbol{x}_t \in \mathbb{R}^n$ is a state vector, $\boldsymbol{u}_t \in \mathbb{R}^m$ is an input vector, $\boldsymbol{y}_k \in \mathbb{R}^l$ is an output vector, $\boldsymbol{\theta} \in \mathbb{R}^p$ is a parameter vector, $\boldsymbol{A}(\cdot) \in \mathbb{R}^{n \times n}$, $\boldsymbol{B}(\cdot) \in \mathbb{R}^{n \times m}$, $\boldsymbol{\sigma}(\cdot) \in \mathbb{R}^{n \times n}$, $\boldsymbol{C}(\cdot) \in \mathbb{R}^{l \times n}$ and $\boldsymbol{D}(\cdot) \in \mathbb{R}^{l \times m}$ are nonlinear functions, $\{\boldsymbol{\omega}_t\}$ is an n-dimensional standard Wiener process and $\{\boldsymbol{e}_k\}$ is an l-dimensional white noise process with $\boldsymbol{e}_k \in N(\boldsymbol{0}, \boldsymbol{S}(\boldsymbol{\theta}))$.

Parameter estimation methods

CTSM-R allows a number of different parameter estimation setups to be used.

Maximum likelihood estimation

Given a particular model structure, *maximum likelihood* (ML) estimation of the unknown parameters can be performed by finding the parameters $\boldsymbol{\theta}$ that maximize the likelihood function of a sequence of measurements $\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k, \ldots, \boldsymbol{y}_N$ of the output variables. By introducing the notation:

$$\mathcal{Y}_k = [\boldsymbol{y}_k, \boldsymbol{y}_{k-1}, \ldots, \boldsymbol{y}_1, \boldsymbol{y}_0] \tag{1.7}$$

the likelihood function is the joint probability density:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = p(\mathcal{Y}_N|\boldsymbol{\theta}) \tag{1.8}$$

or equivalently:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = \left(\prod_{k=1}^{N} p(\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}) \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \tag{1.9}$$

where the rule $P(A \cap B) = P(A|B)P(B)$ has been applied to form a product of conditional probability densities. By introducing the notation:

$$\hat{\boldsymbol{y}}_{k|k-1} = E\{\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}\} \tag{1.10}$$

$$\boldsymbol{R}_{k|k-1} = V\{\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}\} \tag{1.11}$$

and:

$$\boldsymbol{\epsilon}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1} \tag{1.12}$$

and by assuming that the conditional probability densities are Gaussian, the likelihood function can be written as follows:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = \left(\prod_{k=1}^{N} \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1} \boldsymbol{\epsilon}_k\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1})}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \tag{1.13}$$

where $\boldsymbol{\epsilon}_k$ and $\boldsymbol{R}_{k|k-1}$ can be computed by means of a Kalman filter (LTI models) or an iterated extended Kalman filter (LTV and NL models). Further conditioning on $\boldsymbol{y}_0$ and taking the negative logarithm gives:

$$\begin{aligned} -\ln\left(L(\boldsymbol{\theta}; \mathcal{Y}_N|\boldsymbol{y}_0)\right) = \frac{1}{2}\sum_{k=1}^{N}\left(\ln(\det(\boldsymbol{R}_{k|k-1})) + \boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1}\boldsymbol{\epsilon}_k\right) \\ + \frac{1}{2}\left(\sum_{k=1}^{N} l\right)\ln(2\pi) \end{aligned} \tag{1.14}$$

and ML estimates of the parameters (and the initial states if these are unknown) can now be determined by solving the nonlinear optimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta}\left\{-\ln\left(L(\boldsymbol{\theta}; \mathcal{Y}_N|\boldsymbol{y}_0)\right)\right\} \tag{1.15}$$

Maximum a posteriori estimation

If prior information about the parameters is available in the form of a prior probability density function $p(\boldsymbol{\theta})$, Bayes' rule can be applied to give an improved estimate by forming the posterior probability density function:

$$p(\boldsymbol{\theta}|\mathcal{Y}_N) = \frac{p(\mathcal{Y}_N|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{Y}_N)} \propto p(\mathcal{Y}_N|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{1.16}$$

and finding the parameters that maximize this function, i.e. by performing *maximum a posteriori* (MAP) estimation. By introducing the notation:

$$\boldsymbol{\mu}_{\boldsymbol{\theta}} = E\{\boldsymbol{\theta}\} \tag{1.17}$$
$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = V\{\boldsymbol{\theta}\} \tag{1.18}$$

and:

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}} = \boldsymbol{\theta} - \boldsymbol{\mu}_{\boldsymbol{\theta}} \tag{1.19}$$

and by assuming that the prior probability density of the parameters is Gaussian, the posterior probability density function can be written as follows:

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{Y}_N) \propto & \left(\prod_{k=1}^{N} \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1} \boldsymbol{\epsilon}_k\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1})}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \\ & \times \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\right)}{\sqrt{\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})}\left(\sqrt{2\pi}\right)^p} \end{aligned} \tag{1.20}$$

Further conditioning on $\boldsymbol{y}_0$ and taking the negative logarithm gives:

$$\begin{aligned} -\ln\left(p(\boldsymbol{\theta}|\mathcal{Y}_N, \boldsymbol{y}_0)\right) \propto & \frac{1}{2}\sum_{k=1}^{N}\left(\ln(\det(\boldsymbol{R}_{k|k-1})) + \boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1}\boldsymbol{\epsilon}_k\right) \\ & + \frac{1}{2}\left(\left(\sum_{k=1}^{N} l\right) + p\right)\ln(2\pi) \\ & + \frac{1}{2}\ln(\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})) + \frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\epsilon}_{\boldsymbol{\theta}} \end{aligned} \tag{1.21}$$

and MAP estimates of the parameters (and the initial states if these are unknown) can now be determined by solving the nonlinear optimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta}\left\{-\ln\left(p(\boldsymbol{\theta}|\mathcal{Y}_N, \boldsymbol{y}_0)\right)\right\} \tag{1.22}$$

Using several independent data sets

Instead of a single sequence of measurements, several separate sequences, i.e. $\mathcal{Y}_{N_1}^1$, $\mathcal{Y}_{N_2}^2$, …, $\mathcal{Y}_{N_i}^i$, …, $\mathcal{Y}_{N_S}^S$, possibly of varying length, may be available. In

this case a similar estimation method can be applied by expanding the expression for the posterior probability density function to a more general form:

$$p(\boldsymbol{\theta}|\mathbf{Y}) \propto \prod_{i=1}^{S} \left(\prod_{k=1}^{N_i} \frac{\exp\left(-\frac{1}{2}(\boldsymbol{\epsilon}_k^i)^T (\boldsymbol{R}_{k|k-1}^i)^{-1} \boldsymbol{\epsilon}_k^i\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1}^i)}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0^i|\boldsymbol{\theta}) \times \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\right)}{\sqrt{\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})}\left(\sqrt{2\pi}\right)^p} \tag{1.23}$$

where:

$$\mathbf{Y} = [\mathcal{Y}_{N_1}^1, \mathcal{Y}_{N_2}^2, \ldots, \mathcal{Y}_{N_i}^i, \ldots, \mathcal{Y}_{N_S}^S] \tag{1.24}$$

and where the individual sequences of measurements are assumed to be stochastically independent. Further conditioning on:

$$\mathbf{y_0} = [\boldsymbol{y}_0^1, \boldsymbol{y}_0^2, \ldots, \boldsymbol{y}_0^i, \ldots, \boldsymbol{y}_0^S] \tag{1.25}$$

and taking the negative logarithm gives:

$$\begin{aligned} -\ln\left(p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{y_0})\right) \propto & \frac{1}{2} \sum_{i=1}^{S} \sum_{k=1}^{N_i} \left(\ln(\det(\boldsymbol{R}_{k|k-1}^i)) + (\boldsymbol{\epsilon}_k^i)^T (\boldsymbol{R}_{k|k-1}^i)^{-1} \boldsymbol{\epsilon}_k^i \right) \\ & + \frac{1}{2} \left(\left(\sum_{i=1}^{S} \sum_{k=1}^{N_i} l \right) + p \right) \ln(2\pi) \\ & + \frac{1}{2} \ln(\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})) + \frac{1}{2} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}} \end{aligned} \tag{1.26}$$

and estimates of the parameters (and the initial states if these are unknown) can now be determined by solving the nonlinear optimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \left\{ -\ln\left(p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{y_0})\right) \right\} \tag{1.27}$$

Data issues

Within CTSM-R features have also been implemented for dealing with varying sample times, occasional outliers and missing observations.

Statistical tests

In addition to the parameter estimates themselves, CTSM-R computes the standard deviations of the estimates, the corresponding t-test scores and associated probabilities for testing the significance of the parameters, as well as the correlation matrix of the estimates. Altogether these quantities allow a number of statistical tests to be performed to investigate the quality of the model.

Residual analysis

To facilitate residual analysis CTSM-R can be used to generate validation data in the form of state and output predictions corresponding to a given input data set, using either one-step-ahead prediction or pure simulation.

Chapter 2

Installation

First, download the latest version of R from `www.r-project.org` and install it on the computer. Use preferably the newest version and at least a version newer than 2.14. The installation of CTSM-R is carried out through the package system in R. To install CTSM-R in R simply run the following line:

```
## Install the package
install.packages("ctsmr", repo = "http://ctsm.info/repo/dev", type = "source")
```

Note that if this gives an error message then restart R and try again.

2.1 Dependencies

CTSM-R translates the models into Fortran code which has to be compiled at your local machine.

- **Linux** users must install R development packages. Ubuntu users can in a terminal run `sudo apt-get install r-base-dev` which will install all the required packages.
- **Windows** users must install the appropriate version of Rtools from `http://cran.r-project.org/bin/windows/Rtools/`.
- **Mac** users must install either Xcode or the much smaller Command Line Tools for Xcode (requires a free developer account at Apple). Additionally you must install gfortran from `http://cran.r-project.org/bin/macosx/tools/`.

Chapter 3

Using CTSM-R

In the present section all features of CTSM-R are described. If you just want to get started quickly then follow and run the code of one of the examples listed in Section 3.12.

3.1 Create a model

In order to use the CTSM-R package it is must be loaded with

```
library(cstmr)
```

and an object of the class "ctsm" must be generated

```
model <- ctsm$new()
```

The CTSM-R is implemented using ReferenceClasses[1], which provide object oriented programming in R. The fields and methods on the variable `model` can be used for the modeling.

3.2 Rules for model equations

The following rules apply when defining model equations:

- Characters accepted by the interpreter are letters **A-Z** and **a-z**, integers **0-9**, operators **+, -, *, /, ^**, parentheses **(** and **)** and decimal separator **.**.
- The interpreter is case sensitive with respect to the symbolic names of inputs, outputs, states, algebraic equations and parameters, but not with respect to common mathematical functions. This means that the names '**k1**' and '**K1**' are different, whereas the names '**exp()**' and '**EXP()**' are the same. The single character '**t**' is treated as the time variable, whereas the single character '**T**' is treated as any other single character.
- The number formats accepted by the interpreter are the following: Scientific (i.e. **1.2E+1**), standard (i.e. **12.0**) and integer (i.e. **12**).

[1] `http://www.inside-r.org/r-doc/methods/ReferenceClasses`

- Each factor, i.e. each collection of latin letters and integers separated by operators or parentheses, which is not a number or a common mathematical function, is checked to see if it corresponds to the symbolic name of any of the inputs, outputs, states or algebraic equations or to the time variable. If not, the factor is regarded as a parameter.
- The common mathematical functions recognized by the interpreter are the following: **abs()**, **sign()**, **sqrt()**, **exp()**, **log()**, **sin()**, **cos()**, **tan()**, **arcsin()**, **arctan()**, **sinh()** and **cosh()**.

3.3 Defining inputs, outputs, equations and initial values

In CTSM-R the definition of a model is carried out with methods on the model object. The following must be defined for a model before parameter estimation can be carried out:

- The inputs (as named in the data) see Section 3.4 below.
- The states with accompanying system equations.
- The outputs (as named in data) with accompanying observation equations.
- The lower and upper bounds, and initial values of the states and parameters which are to be estimated. The parameters and states can be fixed by only defining their initial value.

Inputs

Defining the names of the inputs is carried out with the function

```
model$addInput(...)
```

where the inputs are simply defined as a comma separated list, for example `model$addInput(U1,U2)` adds the inputs `U1` and `U2`, which must be columns in the data, see Section 3.4.

States and system equations

Adding a state is with its system equation to the model is carried out with the function

```
model$addSystem(dX1 ~ (...) * dt + (...) * dw1)
```

where `X1` is the name of the state, and the `...` are expressions of states and inputs. More system equations can be added with the function by simply calling it again, for example

```
model$addSystem(dX2 ~ (...) * dt + (...) * dw2)
```

adds a second state `X2` with an independent Wiener process `dw2`.

Outputs and observation equations

Adding an observation equation is carried out with the function

```
model$addObs(Y1 ~ ...)
```

where `Y1` is the name of the output (as defined in the data, see Section 3.4) and `...` is an expression of the states and inputs. The variance of the observation noise is added for the output `Y` with the function

```
model$setVariance(Y1 ~ ...)
```

where `...` is an expression of the states and inputs. Adding one more observation equation is simply carried out by calling the functions again, for example

```
model$addObs(Y2 ~ ...)
model$setVariance(Y2 ~ ...)
```

adds the output `Y2` to the model.

Defining initial values of states and parameters

The parameter estimation is carried out by maximizing the likelihood function with an optimization scheme. The scheme requires initial values for the states and parameters, together with lower and upper bounds. The initial values can be set with the function

```
model$setParameter(a = c(init = 10, lower = 1, upper = 100))
```

which sets the initial value and bounds for the parameter `a`. For setting the initial value of a state the same function is used, for example

```
model$setParameter(X1 = c(init = 100, lower = 10, upper = 1000))
```

sets the intial value of `X1`.
In order to fix the value of a parameter to a constant value the function is called without the bounds, for example

```
model$setParameter(a = c(init = 10))
```

and to estimate the parameter with the maximum a posteriori method, see Chapter 1.2, the prior standard deviation is set with

```
model$setParameter(a = c(init = 10, lower = , upper = , sd = ))
```

Note that

3.4 Defining data for estimation

The data is defined as a data.frame, see [2] for description of data frames i R. The columns of the data.frame must be the time series of the time, input(s), and output(s). The columns to be used must have class `numeric` and be named as in the model equations. The time column must be named as `"t"` or `"time"`.

Several independent data sets can be used for estimation by providing a list of data.frames formatted as described above.

3.5 Parameter estimation

The parameter estimation is carried out with the function

```
fit <- model$estimate(data, constantsamplingtime = TRUE, firstorderinputinterpolation = FALSE,
    threads = 1)
```

where

- `data` is the dataframe used for estimation,
- `constantsamplingtime` Set to `TRUE` if the samples are equidistant (i.e. constant sample period) or else set to `FALSE`. Note, that a check is carried and a warning given if a wrong value is set, for example if set to `TRUE` but the time differences are not equal (or near equal as determined with `all.equal()`).
- `firstorderinputinterpolation` Set to `FALSE` for zero order hold (constant input between the sample points) or else set to `TRUE` for 1'st order hold (linear interpolated inputs between the sample points).
- and `threads` is the number of threads to be used for parallel computations. The parameter estimation is carried out by minimizing the objective function in Equation (1.14). The trace of the optimization is printed to the R console during the estimation.

The `fit` returned is a list with information on the result of the estimation. For a complete description of all elements, see `ctsm$help("estimate")`. See all the names of all elements with

```
names(fit)
```

The main elements are:

- `loglik` The loglikelihood of the model with the estimated parameters
- `xm` The estimated values of the parameters
- `sd` The estimated standard deviance of the parameter estimates
- `model` The model used for the estimation. The values of `constantsampletime` and `firstorderinputinterpolate` are also stored with the model in the fit.
- `data` The data used for the estimation.

[2] `cran.r-project.org/manuals.html`

3.6 Displaying estimation results and generating validation data

Several functions are available for displaying the estimation results and for generating validation data when the initial states and parameters have been estimated (see Section 3.11 for an explanation of error messages if the computation stops prematurely).

The importance of validating the estimation results must be emphasized. It is necessary to verify that the optimization did end at a global optimum. The four points marked with red in the text below should be considered, when checking the estimation results. Finally, a validation of the assumption of white noise one-step prediction residuals, together with a physical interpretation of the parameter estimates, should be carried out as outlined in Section 3.7.

Summary of estimated parameters

The function

```
summary(fit, extended = TRUE)
```

takes the `fit` returned by `model$estimate()`. It displays a matrix of the estimated parameters with their estimated uncertainty, some optimization variables and the correlation matrix of the parameter estimates. The displayed results for each parameter are:

`Estimate`	The estimated parameter value.
`Std. Error`	The standard deviance of the parameter estimate.
`t value`	The value of the t-statistic
`Pr(>|t|)`	The fraction of probability of the corresponding t-distribution outside the limits set by the `t value`. Loosely speaking, the p($>\|t\|$) value is the probability that the particular initial state or parameter is insignificant, i.e. equal to 0. ***If this value is not low*** (normally it should be below 0.05) this can be an indication that the model is over-parametrized.
`dF/dPar`	Derivative of the objective function with respect to the particular initial state or parameter. ***If the value is not close to zero***, the solution found may not be the true optimum, and it should be considered to change some settings for the optimization and repeating the computation.
`dPen/dPar`	Derivative of the penalty function with respect to the particular initial state or parameter. ***If the value is significant compared to the `dF/dPar` value***, the particular initial state or parameter may be close to one of its limits, and it should be considered to loosen this limit

Finally, the Correlation Matrix of the parameter estimates is printed. ***If the Correlation Matrix has off-diagonal values close to 1 or -1***, it is an indication that the model is over-parametrized, and it should be considered to eliminate some of the parameters.

Prediction, simulation and filtering

In this section it is described how predictions, simulations and filtering can be carried with CTSM-R. First

Prediction

Prediction data, i.e. state and output predictions j steps ahead based on the inputs and outputs in a given data set, can be generated for a fit of a model, both with the data set used for estimation and for other data sets. The state and output predictions generated are $\hat{\boldsymbol{x}}_{t_k|t_{k-j}}$ and $\hat{\boldsymbol{y}}_{t_k|t_{k-j}}$ for $j \leq 1$ corresponding to each time instant t_k in the data set. Also generated are their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{t_k|t_{k-j}}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{t_k|t_{k-j}})}$ and $\mathrm{SD}(\hat{\boldsymbol{y}}_{t_k|t_{k-j}}) = \sqrt{\mathrm{diag}(\boldsymbol{R}_{t_k|t_{k-j}})}$.

Simulation

Pure simulation data, i.e. state and output predictions based only on the inputs in a given data set, can be generated for a fit of a model. The state and output predictions generated are $\hat{\boldsymbol{x}}_{t_k|t_0}$ and $\hat{\boldsymbol{y}}_{t_k|t_0}$ corresponding to each time instant t_k in the data set. Also generated are their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{t_k|t_0}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{t_k|t_0})}$ and $\mathrm{SD}(\hat{\boldsymbol{y}}_{t_k|t_0}) = \sqrt{\mathrm{diag}(\boldsymbol{R}_{t_k|t_0})}$.

Filtering

Filtered data, i.e. the estimates of the states at time t based on the inputs and outputs available at time t, can be generated for a fit of a model. The state estimates generated are $\hat{\boldsymbol{x}}_{t_k|t_k}$ along with its standard deviation $\mathrm{SD}(\hat{\boldsymbol{x}}_{t_k|t_k}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{t_k|t_k})}$.

Smoothing

Smoothing data, i.e. the estimates of the states at time t based on the entire data set, can be generated for a fit of a nonlinear model. The state estimates generated are $\hat{\boldsymbol{x}}_{t_k|t_N}$ along with their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{t_k|t_N}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{t_k|t_N})}$.

Generating predictions, simulations, filtered and smoothed series

Predictions, simulations, filtering and smoothing with a fit from a model can be generated with the functions

```
Pred <- predict(fit, n.ahead = 1)
Sim <- simulate.ctsmr(fit)
Fil <- filter.ctsmr(fit)
Smo <- smooth.ctsmr(fit)
```

respectively, where `fit` is the list returned by `model$estimate()`. For predictions the `n.ahead` determines the number of steps ahead of the predictions. To generate the series using a different data set than for the estimation the following arguments must also be given to the methods

- `newdata` as a data.frame with the appropriate format for the model (see Section 3.4)
- `constantsamplingtime` set to `TRUE` for equidistant sample points and `FALSE` for varying sample time
- `firstorderinputinterpolation` set to `TRUE` for first order interpolation of the inputs between the sample points (see Section 3.5).

A list is returned with values of the states and the observations, which can accessed by

```
## A matrix of the predicted states and estimated standard
## devations
Pred$state$pred
Pred$state$SD
## A matrix of the predicted observations and estimated
## standard devations
Pred$output$pred
Pred$output$SD
```

and similarly for simulation. For filtering and smoothing only the state is computed. Note that the returned value (above `Pred`) is either

- if the data used was a single data.frame, then `$pred` and `$SD` are matrices
- or if the data was a list of data.frames, then `$pred` and `$SD` are equivalent lists of matrices

3.7 Model validation

To verify that the model and the estimated parameter values describes the dynamics of the system, the following should be considered:

- The assumption of white noise residuals should be inferred upon using the auto-correlation function (ACF) and the cumulated periodogram (CP), which can also reveal how well dynamics on different timescales are modeled. In R functions for carrying out such analysis are implemented, e.g. `acf()` and `cpgram()`.
- Plots of the inputs, outputs, and residuals. These plots can be used to understand which effects are not described properly by the model.
- Evaluation of the estimated physical parameters. Clearly the results should be consistent among different models, for example estimate of the thermal resistance of a building envelope should not change significantly among the models. Furthermore, the modeller have to judge if the results are consistent with reality.

Systematic approaches to grey-box model selection are presented by [3] and [1]. For examples of an implementation of this model validation, see the scripts accompanying the examples listed in Section 3.12.

3.8 Changing the settings for a model

Using the Settings menu, it is possible to control a number of the computational features within CTSM-R, which may be useful to obtain better results. The settings are controlled with the elements in the list

```
model$options$element
```

where the following `elements` can be set.
Filter settings which holds the controls for the (iterated extended) Kalman filter part of CTSM-R (see [2] for details).

- `initialVarianceScaling` For all models the Scaling factor for initial covariance (default: 1.0) can be set. It is used in the calculation of the covariance of the initial states. *The larger the scaling factor, the more uncertain the initial states*.
- `numberOfSubsamples` For linear time varying (LTV) models, the Number of subsamples in iterated extended Kalman filter (default: 10) is displayed in addition to the above scaling factor. This is the number of subsamples used in the subsampling approximation to the true solution of the propagation equations in the iterated extended Kalman filter. *The more subsamples, the more accurate the approximation*.

Non-linear models:

- `solutionMethod` which method to apply for solving the propagation equations of the iterated extended Kalman filter: 1) Subsampling approximation, 2) numerical ODE solution with the Adams method for non-stiff systems (default) or 3) numerical ODE solution with the BDF method for stiff systems.
- `odeeps` In the lower panel the Tolerance for numerical ODE solution (default: 1.0E-12) is the tolerance used by the ODE solvers in the iterated extended Kalman filter. *The lower the tolerance, the more accurate the ODE solution*.
- `nIEKF` The Maximum number of iterations in iterated extended Kalman filter (default: 10) is the maximum number of iterations performed to reduce the effects of measurement equation nonlinearities in the iterated extended Kalman filter. *The more iterations and the lower the tolerance, the better*.
- `iEKFeps` the Tolerance for iterated extended Kalman filter (default: 1.0E-12) is the corresponding tolerance. The measurement equation is iterated until the tolerance is met or until the maximum number of iterations is reached.

Optimization settings holds the basic controls for the optimisation part of CTSM-R (see the Mathematics Guide for details).

- `maxNumberOfEval` The Maximum number of objective function evaluations (default: 500) is simply a limit on the number of objective function evaluations.

- `eps` the Relative error in calculation of objective function (default: 1.0E-14) is a value used to determine a step size for the finite difference gradient approximations within the optimisation algorithm.
- `eta` the Adjustment factor for initial step length in line search (default: 1.0E-6) is a value used to adjust the initial step length in the line search.

Advanced settings (there is usually no need to adjust any of these values):

- `hubersPsiLimit` The Cut-off value for Huber's psi-function (default: 3.0) is the constant c in Huber's ψ-function.
- `padeApproximationOrder` The Padé approximation order (default: 6) is the order of the Padé approximation used to compute matrix exponentials.
- `svdEps` The Tolerance for singular value decomposition (default: 1.0E-12) is a value used to determine if the $\boldsymbol{A}$ matrix is singular or not.
- `lambda` The Lagrange multiplier in penalty function (default: 1.0E-4) is the λ value used in the penalty function.
- `smallestAbsValueForNormalizing` The Minimum absolute value used for normalizing in penalty function is a value used to ensure numerical stability in calculation of the penalty function.

3.9 Save and load a fit

The fit returned by `model$estimate()` can simply be saved with

```
save(fit, file = "xx.rda")
```

and loaded again with

```
load(file = "xx.rda")
```

which recreates `fit` in the environment.

3.10 Additional features of CTSM-R

- Profile likelihood
- Optimization trace

3.11 Error messages

In this section the error messages, which CTSM-R gives if it encounters errors, are listed together with explanations and hints on how to resolve the error. Mainly errors occur when CTSM-R is analyzing the model, e.g. if an equation is not defined correctly, or in the optimization, e.g. if the optimization did not converge.

Estimation

When calling `model$estimate()` to carry out the parameter estimation the following error messages can be returned

- `Missing initial values for a.` where `a` is a name of a variable used in the model. The cause for this error can be
  - The initial values were not defined for parameter `a`
  - `a` is an input which is defined in the equations, but not in `model$addInput()`
  - `a` should not have been defined in the equations at all or given a wrong name

3.12 Modeling examples

Several examples with accompanying R scripts are available. They are located in the folder `examples`. The following examples are available:

- `Building 1` Simple example of modeling the heat dynamics of a building using the heat flux through the building envelope as model output. A linear time invariant model is used.
- `Building 2` Example of modeling the heat dynamics of a building using the indoor temperature as model output. Linear time invariant models are used. It is furthermore demonstrated how a suitable model can be selected using likelihood ratio tests.

Chapter 4

Tips and tricks

In this section tips and tricks for using R and modeling with CTSM-R is given.

4.1 Matlab/R reference

For Matlab users the Matlab/R reference guide provides a very easy way of learning to use R, see `http://www.math.umaine.edu/~hiebeler/comp/matlabR.html`

4.2 Editor for R

Several editors are available for interaction with R.

RStudio

RStudio is a well developed editor for R which has implemented many features such as auto-completion, etc., see more on `rstudio.org`.

RStudio on the DTU Linux system

For running CTSM-R in R on the DTU Linux system with RStudio proceed by:

- Install the Thinlinc client from `http://www.cendio.com/downloads/clients/`.
- Login with Thinlinc client to "thinlinc.gbar.dtu.dk"
- Instructions for running RStudio on the DTU Unix system:
  - Download it from `http://rstudio.org/download/desktop#`. Take under **Zip/Tarball** the `Debian 6+/Ubuntu 10.04+ (64-bit)` version.
  - Extract it to some `"[folder]"` from where you will run it.
  - Open a terminal, execute `"export RSTUDIO_WHICH_R=/appl/R/2.14.0/bin/R"`, and press enter.
  - Open a terminal, run the binary file `"[folder]/bin/rstudio"`.
  - The computers run Linux, therefore install the CTSM-R package for Linux, see Section 2.

- You can ftp to get your files (plots etc.) to `ftp.student.dtu.dk` and login.

Emacs Speak Statistics (ESS)

Emacs users can interact with R with ESS, see `ess.r-project.org`.

4.3 Model debugging

Tips for model debugging and how to find out why the estimation is not successful.

Debugging functions in R

R has several build-in facilities for debugging functions.

`traceback()`

After an error the `tracback()` can be used to find the function which caused the error.

`browser()`

An easy way to debug a function, where the code can be edited and resourced, is to use `browser()`. It must be set into a function, for example

```
hellobrowser <- function(x) {
    ## Print x
    print(x)
    ## Stop excution here
    browser()
    ## Return x
    return(x)
}
```

will stop the execution of the function when it reaches `browser()`. A promt will be presented in the R console where debugging commands can be run.

`debug` package

The `debug` package can be used to debug all functions, also build in R functions, and the CTSM-R methods. Use `install.packages("debug")` to install the package. This package works very well for debugging Reference classes.

Bibliography

[1] Peder Bacher and Henrik Madsen. "Identifying suitable models for the heat dynamics of buildings". In: *Energy and Buildings* 43.7 (2011), pp. 1511–1522. ISSN: 03787788. DOI: 10.1016/j.enbuild.2011.02.005.

[2] Niels Rode Kristensen and Henrik Madsen. *Continuous Time Stochastic Modelling, CTSM 2.3 - Mathematics Guide*. Tech. rep. DTU, 2003.

[3] Niels Rode Kristensen, Henrik Madsen, and Sten Bay Jørgensen. "A method for systematic improvement of stochastic grey-box models". In: *Computers & Chemical Engineering* 28.8 (2004), pp. 1431–1449. ISSN: 0098-1354. DOI: DOI:10.1016/j.compchemeng.2003.10.003.