Continuous Time Stochastic Modeling in R

User's Guide and Reference Manual

CTSM-R Development Team

CTSM-R Version 1.0.0

April 7, 2015

CTSM-R Development Team. 2015. *CTSM-R: User's Guide and Reference Manual*. Version 1.0.0

Preface

Continuous Time Stochastic Modelling for R or CTSM-R is a free, open source and cross platform tool for identifying physical models using real time series data. Visit ctsm.info for all information and how to install CTSM-R on your computer. Download the user guide from ctsm.info/pdfs/userguide.pdf. Both non-linear and non-stationary systems can be modelled using CTSM-R. By using a continuous time formulation of the dynamics and discrete time measurements the framework bridges the gap between physical and statistical modelling.

CTSM-R is the latest incarnation of CTSM which itself can be traced back more than 30 years. CTSM-R extends the free and open source statistical platform R (www.r-project.org) thus combining the power of CTSM and the data handling, plotting and statistical features of R. CTSM was partly developed during the EU PASSYS project (1 April 1986 to 31 March 1989) and a number of follow up research project funded by EU.

CTSM-R has been successfully applied to a wide range of data-driven modelling applications: heat dynamics of walls and buildings, dynamics of heat exchangers, radiators and thermostats, solar thermal collectors, building integrated photovoltaic systems and more.

It is possible to generate both pure simulation and k-step prediction estimates of the states and the outputs, filtered estimates of the states and, for nonlinear models, smoothed estimates of the states. Standard deviations of all state and output estimates are now also automatically provided. The CTSM modelling framework includes methods for model validation and for selecting the best model among a class of candidate models.

How to contribute

The CTSM-R reference manual and source code of CTSM-R are controlled using Git hosted at DTU Compute. The repositories are currently managed by Rune Juhl (ruju@dtu.dk).

Reference manual

To obtain the LaTeX files first clone the Git repository

```
git clone ruju@git.compute.dtu.dk:ctsmr-reference
```

Changes must always first be committed locally before pushing to or pulling from the server. Changes are committed using

```
git commit -a -m "[insert descriptive message here]"
```

New files have to be tracked by

```
git add [file(s)]
```

Files can also be deleted using

```
git rm [file(s)]
```

To ensure no untracked files or additional changes exist use

```
git status
```

If all changes have been committed then go ahead and update your local repository by pulling down possible changes from the remote server

```
git pull
```

Any conflicts have to be handled manually before pushing your local changes to the remote server

```
git push
```

CTSM-R code

Contents

Part I

Introduction

1 Why CTSM-R

CTSM-R is an R package providing a framework for identifying and estimating stochastic grey-box models. A grey-box model consists of a set of stochastic differential equations coupled with a set of discrete time observation equations, which describe the dynamics of a physical system and how it is observed. The grey-box models can include both system and measurement noise, and both nonlinear and nonstationary systems can be modelled using CTSM-R.

1.1 Model Structure

The general model structure used in CTSM-R is state space model

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

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

where 1.1 is a continuous time stochastic differential equation which is the physical description of a system. (1.2) is the discrete time observation of the underlying physical system.

CTSM-R is built to automatically handle linear and non-linear models.

1.2 Likelihood

CTSM-R is built using likelihood theory. The likelihood is a probabilistic measure of how likely a set of parameters are given data and a model. The likelihood of a time series is the joint probability density function (pdf)

$$L(\boldsymbol{\theta}, \mathcal{Y}_N) = \mathrm{p}\left(\mathcal{Y}_N | \boldsymbol{\theta}\right) \,, \tag{1.3}$$

where the likelihood L is the probability density function given θ and a time series $\mathcal{Y}_N$ of N observations. By alternating the parameters the likelihood function changes and the goal is to find those parameter values which maximizes the likelihood function.

CTSM-R computes the likelihood function (or an approximation when required) and uses an optimization method to locate the most probable parameters.

2 Getting started

2.1 Prerequisites

To run CTSM-R and start estimating parameters you must first install a few required tools.

R

CTSM-R requires R version 2.15 or newer to work. The latest version of R is available for download at http://www.r-project.org.

Linux users may have a version of R available in the local package manager. Note this may not be the most recent version of R. CRAN offers binaries for the popular distributions e.g. Ubuntu.

Several interfaces to R are available. A popular choice is RStudio http://www.rstudio.com/.

Toolchains

CTSM-R requires a suitable C and Fortran compiler.

Windows

Windows users must install Rtools which provides the required toolchain.

Rtools is available from http://cran.r-project.org/bin/windows/Rtools/ Download and install a version of Rtools which is compatible with your version of R.

Rtools can be installed without modifying any of the settings during the installation. It is not required to add the Rtools installation path to the PATH environment variable.

Linux

Linux users should use their distribution's package manager to install gcc and gfortran compilers. Ubuntu users can simply install the development version of R: sudo apt-get install r-base-dev. The required compilers will then be installed.

2.2 How to install CTSM-R

To install CTSM-R first open R. Run the following line of code in R.

```
install.packages("ctsmr", repo = "http://ctsm.info/repo/dev")
```

If you installed CTSM-R before installing the toolchain you may have to restart R.

2.3 How to use CTSM-R

To use the CTSM-R package it must be loaded

```
library(ctsmr)
```

Part II

Using CTSM-R

FiXme Fatal: Parameters boundaries

FiXme Fatal: Start state value for multiple data sets

FiXme Fatal: MAP

3 Model object

A CTSM-R model structure follows an object oriented style. The model is build by adding the mathematical equations one by one.

3.1 Initialization

Every model must first be initialized. An empty model object is created and the mathematical equations are subsequently added to the object.

```
model <- ctsm()
```

ctsm() is a generator function which defines the reference class ctsm. The retuned object is an instance of the class ctsm which has a set of methods attached to it. The methods are used to define the model structure and parameter boundaries.

3.2 System equations

The continuous time stochastic differential equations are added to model by calling

```
model$addSystem(formula)
```

formula is the SDE written as a valid R formula, e.g.

FiXme Note: use glossaries for the SDE

```
dX ~ f(X,U,t) * dt + g(U,t) * dwX
```

Valid formulaes are described in 3.5

3.3 Measurement equations

The discrete time measurement equation is added to model by

```
model$addObs(formula)
```

formula is the measurement equation like

```
y ~ h(X,U,t)
```

Valid formulaes are described in 3.5. The variance of the measurement noise is added for the output with the function

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

3.4 Inputs

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

```
model$addInput(symbol)
```

symbol specifies the which variables in the system and measurement equations are external inputs. Example: `model$addInput(U1,U2)` adds the inputs `U1` and `u2`, which must be columns in the data.

3.5 Rules for model equations

{sec equation rules}

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**).
- 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()**.

Defining initial values of states and parameters

The parameter estimation is carried out by CTSM-R 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=1E2, lower=10, upper=1E3) )
```

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, the prior standard deviation is set with

```
model$setParameter( a = c(init=10, lower = 1, upper = 100, sd = 2) )
```

4 Data

Data is required to estimate the parameters in a model. The continuous time formulation allows the data to be sampled irregularly.

4.1 Individual dataset

{sec:individual dataset}

The data required for estimating the CTSM-R model must be given in a data.frame where the variables are named exactly the same as in the model. CTSM-R looks for the output variables and specified inputs in the data.frame.

```
data <- data.frame(X1 = c(1,2,3), X2 = , X3 = , y1 = ,)
```

All inputs must be observed at all time points, but missing observations are allowed. Missing or partly missing observations should be marked with NA in the data.frame.

4.2 Multiple independent datasets

Multiple independent datasets is a set of individual datasets collected in a list. The collection of datasets can be of different length.

```
bigdata <- list(data1, data2, data3)
```

5 Settings

FiXme Fatal: scaled initial covariance

The mathematical methods within CTSM-R can be tuned through a number of settings. The settings are found in the list $options in the ctsm object.

```
model$options$[element]
```

where [element] is any of the following options.

Filter settings which holds the controls for the (iterated extended) Kalman filter part of CTSM-R (see [**Kristensen2003**] for details).

`initialVarianceScaling` `numeric, positive` *Default:* 1.0

For all models the Scaling factor for initial covariance 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` `integer, positive` *Default:* 10

For linear time varying (LTV) models, the Number of subsamples in Kalman filter 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 Kalman filter. *The more subsamples, the more accurate the approximation*.

5.1 Non-linear models

`odeeps` `numeric, positive` *Default:* 1.0×10^{-12}

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` `integer, positive` *Default:* 10

The maximum number of iterations in the iterated extended Kalman filter.

`iEKFeps` `numeric, positive` *Default:* 1.0×10^{-12}

The tolerance for the iterated extended Kalman filter. The measurement equation is iterated until the tolerance is met or until the maximum number of iterations is reached.

5.2 Optimization

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

`maxNumberOfEval` `integer, positive` *Default:* 500

The maximum number of objective function evaluations.

`eps` `numeric, positive` *Default:* 1.0×10^{-14}

The relative convergence tolerance (stopping criteria).

`eta` `numeric, positive` *Default:* 1.0×10^{-6}

The adjustment factor for initial step length in the line search.

5.3 Advanced options

There is usually no need to adjust any of these values.

`hubersPsiLimit` `numeric, positive` *Default:* 3.0

The cut-off value for Huber's psi-function is the constant c in Huber's ψ-function.

`padeApproximationOrder` `integer, positive` *Default:* 6

The Padé approximation order used to compute matrix exponentials.

`svdEps` `numeric, positive` *Default:* 1.0×10^{-12}

The tolerance for the singular value decomposition used to determine if the $\backslash A$ matrix is singular or not.

`lambda` `numeric, positive` *Default:* 1.0×10^{-4}

The Lagrange multiplier in the penalty function.

`smallestAbsValueForNormalizing` `numeric, positive` *Default:* NaN

The minimum absolute value used for normalizing in penalty function.

6 Result object

The object returned from $estimate is an S3 ctsmr object. That is a list which contains various elements such as the estimated coefficients ($xm). All parts of the list can be referenced using the $ operator followed by the name of the element.

All elements and their data types are listed below.

`itr` `integer, positive`

The interations to convergence.

`neval` `integer, positive`

The number of function evaluations.

`detH` `numeric`

Determinant of the Hessian of the objective function.

`f` `numeric`

The objective function at convergence.

`fpen` `numeric`

The value of the penalty function at convergence.

`fprior` `numeric`

`loglik` `numeric`

The logarithm of the likelihood function.

`dL`

`dpen` `numeric vector`

Derivative of the penality function wrt. the parameters.

`corr` `numeric matrix`

The correlation matrix of the parameter estimates.

`pt` `numeric vector`

`sd` `numeric vector`

The standard deviance of the parameter estimates.

`t` `numeric vector`

The t-test statistics.

`xm` `named numeric vector`

The parameter estimates.

`xmin` `named numeric vector`

The lower box boundary of the paramters for the optimization.

`xmax` `named numeric vector`

The upper box boundary of the paramters for the optimization.

`estimated`

`trace` `numeric matrix`

Trace of parameters and derivatives during the optimization.

`threads` `integer, positive`

The number of threads used while calculating the gradients.

`cpus` `integer, positive`

The number of available CPU's (cores).

`info` `integer, positive`

An information code.

`message` `character string`

Description of the information code.

`model` `ctsmr`

The CTSM-R model object.

`data` `data.frame`

The data used for the parameter estimation.

6.1 Information codes

A code is always returned from CTSM-R during mathematical operations such as estimation or prediction. Table 6.1 lists the posible codes.

0	Converged.
-1	Terminated.
2	The maximum number of objective function evaluations has been exceeded.
5	The prior covariance matrix is not positive definite.
10	The amount of data available is insufficient to perform the estimation.
20	The maximum objective function value (1×10^{300}) has been exceeded.
30	The state covariance matrix is not positive definite.
40	The measurement noise covariance matrix is not positive definite.
50	Unable to calculate matrix exponential.
60	Unable to determine reciprocal condition number.
70	Unable to compute singular value decomposition.
80	Unable to solve system of linear equations.
90	Unable to perform numerical ODE solution.

Table 6.1: Return codes from CTSM-R

{tbl: return codes}

7 Functions

7.1 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 together with some optimization variables and correlation matrix of the parameter estimates. The displayed results for each parameter are:

`Estimate`	The estimated parameter value.
`Std. Error`	The uncertainty of the parameter estimate.
`t value`	Skal vi ikke fjerne den fra resultatet??
`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 indicate 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 you should consider changing 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 you should consider to loosen this limit
More	If MAP estimation was used...

7.2 Prediction

k-step predictions are computed using the predict function.

```
predict(fit, n.ahead, covariance, newdata, firstorderinputinterpolation, x0, vx0)
```

fit is a ctsmr object returned from the $estimate.
n.ahead is an non-negative integer (default: n.ahead = 1L).
covariance determines if the full covariance matrix is returned. Default is FALSE.
newdata is a data.frame with data as described in FiXme Fatal: reference to the data section
firstorderinputinterpolation if FALSE (default) the inputs are constant between samples. If TRUE linear interpolation is used.
x0 is a numeric vector with initial values to override those in the fit.
vx0 specifies the initial covariance matrix of the states.

7.3 Simulation of mean

Deterministic simulations are computed using the simulate function.

```
simulate(fit, newdata)
```

fit is a ctsmr object returned from the $estimate.
newdata is a data.frame with data as described in FiXme Fatal: reference to the data section

7.4 Simulation (stochastic)

Stochastic realizations of the SDE can be generated. For linear models the densities are known and can be sampled correctly. This is not implemented directly.

For non-linear models stochastic realizations rely on discretization schemes. Realizations are generated using an Euler scheme using

```
stochastic.simulate(fit, x0, data, dt)
```

fit is a ctsmr object returned from the $estimate.
x0 is a numeric vector with initial values of the states.
data is a data.frame with possible inputs.
dt is the discretization step.

7.5 Filtration

Filtration are computed using the filter.ctsmr function.

```
filter.ctsmr(fit, newdata)
```

fit is a ctsmr object returned from the $estimate.
newdata is a data.frame with data as described in FiXme Fatal: reference to the data section

7.6 Smoothing

Smoothed states are computed using the smooth.ctsmr function.

```
smooth(fit, newdata)
```

fit is a ctsmr object returned from the $estimate.

FiXme Fatal: reference to the data section

newdata is a data.frame with data as described in

8 Example

{ex:ex1}

As an example to illustrate the methods we will use a small simulation example (see Figure **??**). A linear 3 compartment model [**lv˙pharmacokinetics˙2013**] similar to the real data modeling example presented in Section **??** is used. We can think of the response (y) as insulin concentration in the blood of a patient, and the input (u) as meals.

The data is simulated according to the model

$$d\boldsymbol{x}_t = \left(\begin{bmatrix} u_t \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -k_a & 0 & 0 \\ k_a & -k_a & 0 \\ 0 & k_a & -k_e \end{bmatrix} \boldsymbol{x}_t \right) dt + \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} d\boldsymbol{\omega}_t \tag{8.1}$$ {eq:exsde1}

$$y_k = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}_{t_k} + e_k \,, \tag{8.2}$$ {eq:exobs1}

where $\boldsymbol{x} \in \mathbb{R}^3$, $e_k \sim \mathcal{N}(0, s^2)$, $t_k = \{1, 11, 21, ...\}$ and the specific parameters ($\boldsymbol{\theta}$) used for simulation are given in Table 8.1 (first column).

The structure of the model (8.1) will of course usually be hidden and we will have to identify the structure based on the measurements as given in Figure **??**. As a general principle simple models are preferred over more complex models, and therefore a first hypothesis could be (Model 1)

$$dx_t = (u_t - k_e x_t)\, dt + \sigma_3 d\omega_t \tag{8.3}$$ {eq:exsde2}

$$y_k = x_{t_k} + e_k \,. \tag{8.4}$$ {eq:exobs2}

As noted above a first approach to model the data could be a 1-state model (Equations (8.3)-(8.4)). The result of the estimation ($\hat{\theta}_1$) is given in Table 8.1, the initial value of the state (x_{30}) and the time constant ($1/k_e$) are both captured quite well, while the uncertainty parameters are way off, the diffusion is too large and the observation variance is too small (with extremely large uncertainty).

The parameters in the model are all assumed to be greater than zero, and it is therefore advisable to estimate parameters in the log-domain, and then transform back to the original domain before presenting the estimates. The log-domain estimation is also the explanation for the non-symmetric confidence intervals in Table 8.1, the confidence intervals are all based on the Hessian of the likelihood at the optimal parameter values, and confidence intervals are based on the Wald confidence interval in the transformed (log) domain. Such intervals could be refined by using profile likelihood based confidence intervals (see also Section **??**).

In order to validate the model and suggest further development, we should inspect the innovation error. When the model is not time homogeneous, the

tab:simex
Table 8.1: Parameter estimates from simulation example, confidence intervals for the individual parameters are given in parenthesis below the estimates. Last two rows present the log-likelihood and the number of degrees of freedom.

	θ	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
x_{10}	40.000	-		38.819
				(29.172,48.466)
x_{20}	35.000	-	107.960	33.421
			(75.211,140.710)	(29.778,37.064)
x_{30}	11.000	10.657	10.641	10.604
		(6.606,14.708)	(10.392,10.889)	(10.281,10.927)
k_a	0.025	-	0.006	0.026
			(0.0038.00778)	(0.025,0.027)
k_e	0.080	0.081	0.056	0.080
		(0.071,0.094)	(0.0418,0.0743)	(0.078,0.083)
σ_1	1.000	-	-	0.5500
				(0.224,1.353)
σ_2	0.200	-	3.616	0.282
			(2.670,4.898)	(0.113,0.704)
σ_3	0.050	2.206	0.001	0.001
		(1.848,2.634)	$(2 \cdot 10^{-55}, 3 \cdot 10^{48})$	$(9 \cdot 10^{-56}, 1 \cdot 10^{49})$
s	0.025	0.0002	0.016	0.031
		$(2 \cdot 10^{-33}, 2.6 \cdot 10^{25})$	(0.0065, 0.0388)	(0.020, 0.049)
$l(\hat{\theta}, \vec{y})$	-	-343.68	-67.85	-19.70
df	-	4	7	9

standard error of the prediction will not be constant and the innovation error should be standardized

$$r_k = \frac{\epsilon_k}{\sqrt{\Sigma_{k|k-1}}}, \tag{8.5}$$ eq:standardres

where the innovation error (ϵ_k) is given in (**??**). All numbers needed to calculate the standardized residuals can be obtained directly from CTSM-R using the function predict. Both the autocorrelation and partial autocorrelation (Figure 8.1) are significant in lag 1 and 2. This suggests a 2-state model for the innovation error, and hence a 3-state model should be used. Consequently we can go directly from the 1-state model to the true structure (a 3-state model).

Now we have assumed that a number of the parameters are actually zero, in a real life situation we might test these parameters using likelihood ratio tests, or indeed identify them through engineering principles. The parameter estimates are given in Table 8.1 ($\hat{\theta}_3$), in this case the diffusion parameter (σ_3) has an extremely wide confidence interval, and it could be checked if this parameters should indeed be zero (again using likelihood ratio test), but for now we will proceed with the residual analysis which is an important part of model validation (see e.g. [**madsen2008**]). The autocorrelation and partial autocorrelation for the 3-state model is shown in Figure 8.2. We see that there are no values outside the 95% confidence interval, and we can conclude that

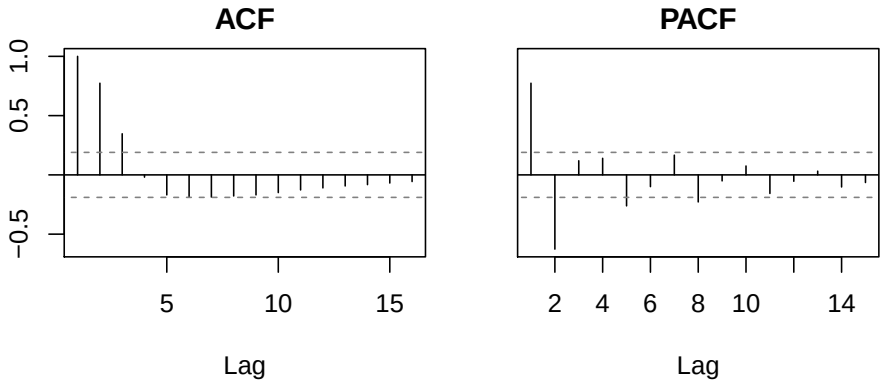

Figure 8.1: Autocorrelation and partial autocorrelation from a simple (1 state) model. `fig:acfplot1`

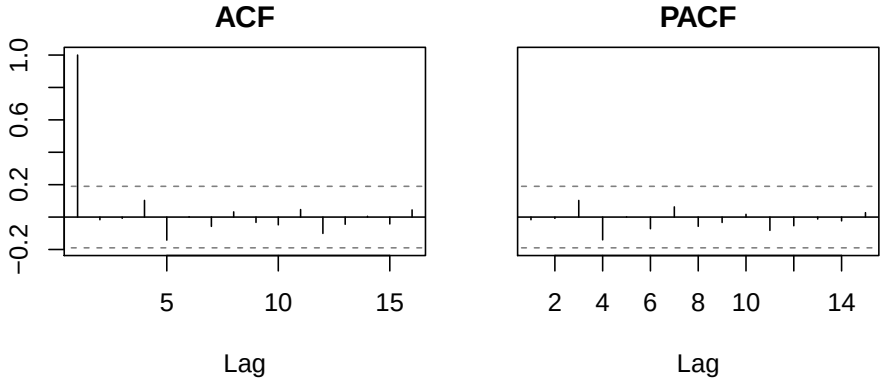

Figure 8.2: Autocorrelation and partial autocorrelation from the 3-state model (i.e. the correct model). `fig:acfplot3`

there is no evidence against the hypothesis of white noise residuals, i.e. the model sufficiently describes the data.

Autocorrelation and partial autocorrelations are based on short-term predictions (in this case 10 minutes) and hence we check the local behavior of the model. Depending on the application of the model we might be interested in longer term behavior of the model. Prediction can be made on any horizon using CTSM-R. In particular we can compare deterministic simulation in CTSM-R (meaning conditioning only on the initial value of the states). Such a simulation plot is shown in Figure 8.3, here we compare a 2-state model (see Table 8.1) with the true 3-state model. It is quite evident that Model 2 is not suited for simulation, with the global structure being completely off, while "simulation" with a 3-state model (with the true structure, but estimated parameters), gives narrow and reasonable simulation intervals. In the case of linear SDE-models with linear observation, this "simulation" is exact, but for

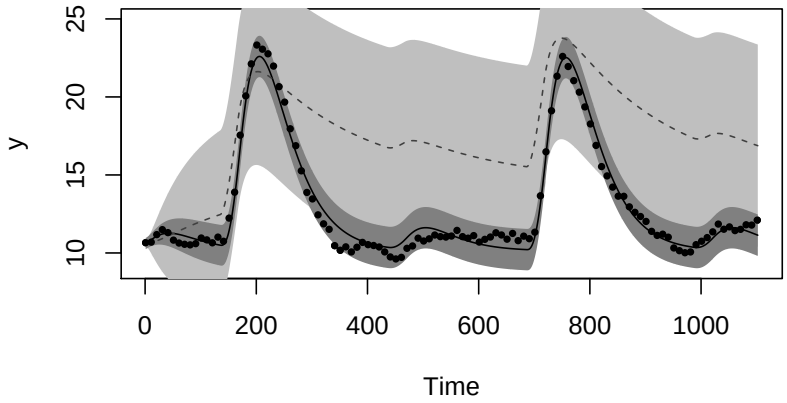

Figure 8.3: Simulation with Model 2 and 3, dashed gray line: expectatoin of Model 2, black line: expectation of Model 3, light gray area: 95% prediction interval for Model 2, dark gray area: 95% prediction interval for Model 3, and black dots are the observations. fig:simModel1and2

nonlinear models it is recommended to use real simulations e.g. using a Euler scheme.

The step from a 2-state model (had we initialized our model development with a 2-state model) to the 3-state model is not at all trivial. It is however clear that model 2 will not be well suited for simulations. Also the likelihood ratio test (or AIC/BIC) supports that Model 3 is far better than Model 2, further it would be reasonable to fix σ_3 at zero (in practice a very small number).

9 Advanced usage

The strength of CTSM-R in R.

9.1 Profile likelihood

9.2 Stochastic Partial Differential Equation

Solar power production of a large PV facility was modelled using a stochastic partial differential equation. The SPDE was discretized into a set of coupled stochastic ordinary differential equations. The model is a linear model with time input dependent transition matrix ($A(\theta, u)$). Normally CTSM-R would treat such a model as a non-linear model, but a modified version was used here to enfore the linear model structure. The data consisted of 12 independent time series and thus the loglikelihood was further parallelized. For further details see [**emil˙jasa˙2015**].

9.3 Population modeling

See [**juhl˙**]

Part III

Mathematical Details

10 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 given sequence of measurements $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, $\ldots$, $\boldsymbol{y}_k$, $\ldots$, $\boldsymbol{y}_N$. By introducing the notation:

$$\mathcal{Y}_k = [\boldsymbol{y}_k, \boldsymbol{y}_{k-1}, \ldots, \boldsymbol{y}_1, \boldsymbol{y}_0] \tag{10.1}$$

the likelihood function is the joint probability density:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = p(\mathcal{Y}_N|\boldsymbol{\theta}) \tag{10.2}$$

or equivalently:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = \left(\prod_{k=1}^{N} p(\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}) \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \tag{10.3}$$

where the rule $P(A \cap B) = P(A|B)P(B)$ has been applied to form a product of conditional probability densities. In order to obtain an exact evaluation of the likelihood function, the initial probability density $p(\boldsymbol{y}_0|\boldsymbol{\theta})$ must be known and all subsequent conditional densities must be determined by successively solving Kolmogorov's forward equation and applying Bayes' rule [**Jazwinski70**], but this approach is computationally infeasible in practice. However, since the diffusion terms in the above model structures do not depend on the state variables, a simpler alternative can be used. More specifically, a method based on Kalman filtering can be applied for linear time invariant (LTI) and linear time varying (LTV) models, and an approximate method based on extended Kalman filtering can be applied for nonlinear models. The latter approximation can be applied, because the stochastic differential equations considered are driven by Wiener processes, and because increments of a Wiener process are Gaussian, which makes it reasonable to assume, under some regularity conditions, that the conditional densities can be well approximated by Gaussian densities. The Gaussian density is completely characterized by its mean and covariance, so by introducing the notation:

$$\hat{\boldsymbol{y}}_{k|k-1} = E[\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}] \tag{10.4}$$

$$\boldsymbol{R}_{k|k-1} = V[\boldsymbol{y}_k|\mathcal{Y}_{k-1}, \boldsymbol{\theta}] \tag{10.5}$$

and:

$$\boldsymbol{\epsilon}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1} \tag{10.6}$$

the likelihood function can be written as follows:

$$L(\boldsymbol{\theta}; \mathcal{Y}_N) = \left(\prod_{k=1}^{N} \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1} \boldsymbol{\epsilon}_k\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1})}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \tag{10.7}$$

where, for given parameters and initial states, $\boldsymbol{\epsilon}_k$ and $\boldsymbol{R}_{k|k-1}$ can be computed by means of a Kalman filter (LTI and LTV models) or an extended Kalman filter (NL models) as shown in Sections 11.1 and 11.2 respectively. Further conditioning on $\boldsymbol{y}_0$ and taking the negative logarithm gives:

$$\begin{aligned} -\ln\left(L(\boldsymbol{\theta}; \mathcal{Y}_N|\boldsymbol{y}_0)\right) = \frac{1}{2}\sum_{k=1}^{N}\left(\ln(\det(\boldsymbol{R}_{k|k-1})) + \boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1}\boldsymbol{\epsilon}_k\right) \\ + \frac{1}{2}\left(\sum_{k=1}^{N} l\right)\ln(2\pi) \end{aligned} \tag{10.8}$$

and ML estimates of the parameters (and optionally of the initial states) can now be determined by solving the following nonlinear optimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \left\{-\ln\left(L(\boldsymbol{\theta}; \mathcal{Y}_N|\boldsymbol{y}_0)\right)\right\} \tag{10.9}$$

11 Kalman Filters

The general idea of the Kalman filter methods used by CTSM, is the calculation of conditional second order moments. For linear systems the filter is exact (except possibly errors due to numerial integration). For nonlinear systems the filter is an approximation (linearizations).

In this part we will use the following short hand notation for the conditional second order moment representation

$$\hat{\boldsymbol{x}}_{t|k} = E[\boldsymbol{X}_t|\mathcal{Y}_{t_k}, \boldsymbol{\theta}] \tag{11.1}$$
$$\hat{\boldsymbol{x}}_{l|k} = E[\boldsymbol{X}_{t_l}|\mathcal{Y}_{t_k}, \boldsymbol{\theta}] \tag{11.2}$$
$$\boldsymbol{P}_{t|k} = V[\boldsymbol{X}_t|\mathcal{Y}_{t_k}, \boldsymbol{\theta}] \tag{11.3}$$
$$\boldsymbol{P}_{l|k} = V[\boldsymbol{X}_{t_l}|\mathcal{Y}_{t_k}, \boldsymbol{\theta}] \tag{11.4}$$
$$\boldsymbol{R}_{l|k} = V[\boldsymbol{Y}_{t_l}|\mathcal{Y}_{t_k}, \boldsymbol{\theta}] \tag{11.5}$$

where t is continous time, t_k is the time of measurement number k, and $\mathcal{Y}_{t_k} = [\boldsymbol{y}_1, ..., \boldsymbol{y}_k]$ (and $\boldsymbol{y}_k = \boldsymbol{y}_{t_k}$).

11.1 Linear Filter

{secKF}

The simplest continuous-discrete time stochastic model is the linear time invariant model, we write it in it's most general form

$$d\boldsymbol{x}_t = (\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t)dt + \sigma d\boldsymbol{w}_t \tag{11.6}$$ {eqKalmanState1}
$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_{t_k} + \boldsymbol{D}\boldsymbol{u}_k + \boldsymbol{e}_k; \quad \boldsymbol{e}_k \sim N(\boldsymbol{0}, \boldsymbol{S}_k) \tag{11.7}$$ {eqKalmanObs1}

where $\boldsymbol{S}_k = \boldsymbol{S}(\boldsymbol{u}_k, t_k)$. We see that even though the model is named linear time invariant, it is allowed to depend on time, but only in the specific way indicated in (11.6)-(11.7).

For the linear time invariant models, $\boldsymbol{\epsilon}_k$ and $\boldsymbol{R}_{k|k-1}$ can be computed for a given set of parameters $\boldsymbol{\theta}$ and initial states $\boldsymbol{x}_0$ by means of a continuous-discrete Kalman filter.

{the:LinKf}

Theorem 1 (Continuous-discrete time Linear Kalman filter) *With given initial conditions $\hat{\boldsymbol{x}}_{1|0} = \boldsymbol{x}_0$, and $\boldsymbol{P}_{1|0} = \boldsymbol{V}_0$, the linear Kalman filter is given by the output* prediction *equations:*

$$\hat{\boldsymbol{y}}_{k|k-1} = \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{D}\boldsymbol{u}_k \tag{11.8}$$
$$\boldsymbol{R}_{k|k-1} = \boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{S}_k \tag{11.9}$$

the innovation *equation:*

$$\boldsymbol{\epsilon}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1} \tag{11.10}$$

the Kalman gain *equation:*

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{R}_{k|k-1}^{-1} \tag{11.11}$$

the updating *equations:*

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\boldsymbol{\epsilon}_k \tag{11.12}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{R}_{k|k-1}\boldsymbol{K}_k^T \tag{11.13}$$

and the state prediction *equations:*

{eqStatePred1}
$$\frac{d\hat{\boldsymbol{x}}_{t|k}}{dt} = \boldsymbol{A}\hat{\boldsymbol{x}}_{t|k} + \boldsymbol{B}\boldsymbol{u}_t \ , \ t \in [t_k, t_{k+1}[\tag{11.14}$$

{eqStatePred2}
$$\frac{d\boldsymbol{P}_{t|k}}{dt} = \boldsymbol{A}\boldsymbol{P}_{t|k} + \boldsymbol{P}_{t|k}\boldsymbol{A}^T + \boldsymbol{\sigma}\boldsymbol{\sigma}^T \ , \ t \in [t_k, t_{k+1}[\tag{11.15}$$

where the following shorthand notation applies

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{A}(\boldsymbol{\theta}) \ , \ \boldsymbol{B} = \boldsymbol{B}(\boldsymbol{\theta}) \\ \boldsymbol{C} = \boldsymbol{C}(\boldsymbol{\theta}) \ , \ \boldsymbol{D} = \boldsymbol{D}(\boldsymbol{\theta}) \\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{\theta}) \ , \ \boldsymbol{S} = \boldsymbol{S}(\boldsymbol{\theta}) \end{aligned} \tag{11.16}$$

Initial conditions ($\hat{\boldsymbol{x}}_{t|t_0} = \boldsymbol{x}_0$ and $\boldsymbol{P}_{t|t_0} = \boldsymbol{P}_0$), for the Kalman filter may either be pre-specified or estimated along with the parameters as part of the overall problem.

In order to show that the Kalman filter holds for linear time invariant systems consider the initial value problem

{eqKalmanState2}
$$d\boldsymbol{x}_t = (\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t)dt + \boldsymbol{\sigma}d\boldsymbol{w}_t \tag{11.17}$$

$$E[\boldsymbol{X}_0] = \boldsymbol{x}_0 \tag{11.18}$$

$$V[\boldsymbol{X}_0] = \boldsymbol{V}_0. \tag{11.19}$$

Now consider the transformation

{eq:KFStrans}
$$\boldsymbol{Z}_t = e^{-\boldsymbol{A}t}\boldsymbol{X}_t \, , \tag{11.20}$$

then by Itô's Lemma, it can be shown that the process $\boldsymbol{Z}_t$ is governed by the Itô stochastic differential equation

{eq:KFZ1}
$$d\boldsymbol{Z}_t = e^{-\boldsymbol{A}t}\boldsymbol{B}(t)dt + e^{-\boldsymbol{A}t}\boldsymbol{\sigma}d\boldsymbol{\omega}_t \tag{11.21}$$

with initial conditions equal the initial conditions of $\boldsymbol{X}_t$. The solution to (11.21) is given by the integral equation

{eq:EKFZ2}
$$\boldsymbol{Z}_t = \boldsymbol{Z}_0 + \int_0^t e^{-\boldsymbol{A}t}\boldsymbol{B}(s)ds + \int_0^t e^{-\boldsymbol{A}s}\boldsymbol{\sigma}(s)d\boldsymbol{\omega}_s \tag{11.22}$$

now the expectation of Z_t is given by

$$E[Z_t] = E[\boldsymbol{Z}_0] + \int_0^t e^{-\boldsymbol{A}t}\boldsymbol{B}(s)ds \tag{11.23}$$

and the variance is

$$V[\boldsymbol{Z}_t] = V[\boldsymbol{Z}_0] + V\left[\int_0^t e^{-As}\boldsymbol{\sigma}d\boldsymbol{\omega}_s\right] \tag{11.24}$$

$$= V[\boldsymbol{Z}_0] + E\left[\left(\int_0^t e^{-As}\boldsymbol{\sigma}d\boldsymbol{\omega}_s\right)\left(\int_0^t e^{-As}\boldsymbol{\sigma}d\boldsymbol{\omega}_s\right)^T\right] \tag{11.25}$$

$$= V[\boldsymbol{Z}_0] + \int_0^t e^{-As}\boldsymbol{\sigma}\boldsymbol{\sigma}^T e^{-A^T s}ds, \tag{11.26}$$

where we have used Itô isometri to get the last equation. Now do the inverse transformation to get the second order moment representation

$$\hat{\boldsymbol{x}}_{t|0} = e^{\boldsymbol{A}t}\hat{\boldsymbol{x}}_0 + \int_0^t e^{\boldsymbol{A}t}\boldsymbol{B}\boldsymbol{u}_s ds \tag{11.27}$$ {eqStatePred3}

$$\boldsymbol{P}_{t|0} = e^{\boldsymbol{A}t}\boldsymbol{V}_0 e^{\boldsymbol{A}^T t} + e^{\boldsymbol{A}t}\int_0^t e^{-\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T e^{-\boldsymbol{A}^T s}ds e^{\boldsymbol{A}^T t} \tag{11.28}$$ {eqStatePred4}

to get the differential formulation given in Theorem 1, differentiate (11.27) and (11.28) with respect to time. In each step the initial values $\boldsymbol{x}_0$ and $\boldsymbol{V}_0$ are replaced by the filter estimates $\boldsymbol{x}_{k|k}$ and $\boldsymbol{P}_{k|k}$.

Input interpolation

The input will only be given at discrete time points and the value of $\boldsymbol{u}_s$ ($s \in (t_k, t_{k+1})$) is calculated by

$$\boldsymbol{u}_s = \begin{cases} \boldsymbol{u}_k & ; \text{ zero order hold} \\ \frac{\boldsymbol{u}_{k+1}-\boldsymbol{u}_k}{t_{k+1}-t_k}(s-t_k) + \boldsymbol{u}_k & ; \text{ first order hold} \end{cases} \tag{11.29}$$

Evaluation of the integrals

Efficient evaluation of the integrals over the matrix exponentials given in (11.27) and (11.28) are by no means a trivial task, but only the principles are given here, while more specific computational issues are given in Chapter 17.

11.2 Extended Kalman Filter

{secEKF}

For non-linear models the innovation error vectors ($\boldsymbol{\epsilon}_k$) and their covariance matrices $\boldsymbol{R}_{k|k-1}$ can be computed (approximated) recursively by means of the Extended Kalman Filter (EKF) as outlined in the following.

Consider first the linear time-varying model

$$d\boldsymbol{X}_t = \left(\boldsymbol{A}(\boldsymbol{u}_t, t, \boldsymbol{\theta})\boldsymbol{X}_t + B(\boldsymbol{u}_t, t, \boldsymbol{\theta})\right)dt + \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta})d\boldsymbol{\omega}_t \tag{11.30}$$ {eq:ConEKFSys1}

$$\boldsymbol{Y}_k = \boldsymbol{C}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta})\boldsymbol{X}_k + \boldsymbol{e}_k, \tag{11.31}$$ {eq:ConEKFObs1}

in the following we will use $\boldsymbol{A}(t)$, $\boldsymbol{B}(t)$, and $\boldsymbol{\sigma}(t)$ as short hand notation for $\boldsymbol{A}(\boldsymbol{u}_t, t, \boldsymbol{\theta})$, $B(\boldsymbol{u}_t, t, \boldsymbol{\theta})$, and $\boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta})$.

We will restrict ourselves to the initial value problem; solve (11.30) (for $t \in [t_k, t_{k+1})$ given that the initial condition $X_{t_k} \sim N(\hat{\boldsymbol{x}}_{k|k}, \boldsymbol{P}_{k|k})$. This is the kind of solution we would get from the ordinary Kalman Filter in the update step.

The linear time-varying system in (11.30)-(11.31) will be used to approximate nonlinear system in the section, but the derivations below will also show how linear time-varying systems can be solved. For linear timevarying systems the Kalman filter equations are still exact. We treat them here as nonlinear systems because they are formally treated as nonlinear systems by CTSM-R. However the linearizations given below is then not approximations but exact, but the exponential integrals are evaluated by forward integration of the ODE given in the Kalman Filter (just as for the nonlinear systems).

Now if we consider the transformation

{eq:EKStrans}
$$\boldsymbol{Z}_t = e^{-\int_{t_k}^t \boldsymbol{A}(s)ds} \boldsymbol{X}_t \tag{11.32}$$

then by Itô's Lemma, it can be shown that the process $\boldsymbol{Z}_t$ is governed by the Itô stochastic differential equation

{eq:EKFZ1}
$$d\boldsymbol{Z}_t = e^{-\int_{t_k}^t \boldsymbol{A}(s)ds} \boldsymbol{B}(t)dt + e^{-\int_{t_k}^t \boldsymbol{A}(s)ds} \boldsymbol{\sigma}(t)d\boldsymbol{\omega}_t \tag{11.33}$$

with initial conditions $\boldsymbol{Z}_{t_k} \sim N(\hat{\boldsymbol{x}}_{k|k}, \boldsymbol{P}_{k|k})$. The solution to (11.33) is given by the integral equation

{eq:EKFZ2}
$$\boldsymbol{Z}_t = \boldsymbol{Z}_{t_k} + \int_{t_k}^t e^{-\int_{t_k}^u \boldsymbol{A}(u)du} \boldsymbol{B}(s)ds + \int_{t_k}^t e^{-\int_{t_k}^s \boldsymbol{A}(u)du} \boldsymbol{\sigma}(s)d\boldsymbol{\omega}_s \tag{11.34}$$

Now inserting the inverse of the transformation gives

{eq:EKFZ3}
$$\begin{aligned} \boldsymbol{X}_t = e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \boldsymbol{X}_0 + e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \int_{t_k}^t e^{-\int_{t_k}^u \boldsymbol{A}(u)du} \boldsymbol{B}(s)ds + \\ e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \int_{t_k}^t e^{-\int_{t_k}^s \boldsymbol{A}(u)du} \boldsymbol{\sigma}(s)d\boldsymbol{\omega}_s \end{aligned} \tag{11.35}$$

Taking the exception and variance on both sides of (11.35) gives

{eq:EKFEx}
$$\hat{\boldsymbol{x}}_{t|k} = e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \hat{\boldsymbol{x}}_{t|k} + e^{-\int_{t_k}^t \boldsymbol{A}(s)ds} \int_{t_k}^t e^{-\int_{t_k}^u \boldsymbol{A}(u)du} \boldsymbol{B}(s)ds \tag{11.36}$$

$$\begin{aligned} \boldsymbol{P}_{t|k} &= e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \boldsymbol{P}_{t|k} e^{\int_{t_k}^t \boldsymbol{A}(s)^T ds} + \\ &\quad e^{\int_{t_k}^t \boldsymbol{A}(s)ds} V\left[\int_{t_k}^t e^{-\int_{t_k}^s \boldsymbol{A}(u)du} \boldsymbol{\sigma}(s)d\boldsymbol{\omega}_s\right] e^{\int_{t_k}^t A(s)^T ds} \\ &= e^{\int_{t_k}^t \boldsymbol{A}(s)ds} V[\boldsymbol{X}_0] e^{\int_{t_k}^t \boldsymbol{A}(s)^T ds} + \end{aligned} \tag{11.37}$$

{eq:EKFVx}
$$e^{\int_{t_k}^t \boldsymbol{A}(s)ds} \int_{t_k}^t e^{-\int_{t_k}^s \boldsymbol{A}(u)du} \boldsymbol{\sigma}(s)\boldsymbol{\sigma}(s)^T e^{-\int_{t_k}^s \boldsymbol{A}(u)^T du} ds e^{\int_{t_k}^t \boldsymbol{A}^T(s)ds} \tag{11.38}$$

where we have used Itô isometry in the second equation for the variance. Now differentiation the above expression with respect to time gives

$$\frac{d\hat{\boldsymbol{x}}_{t|k}}{dt} = \boldsymbol{A}(t)\hat{\boldsymbol{x}}_{t|k} + \boldsymbol{B}(t) \tag{11.39}$$

$$\frac{d\boldsymbol{P}_{t|k}}{dt} = \boldsymbol{A}(t)\boldsymbol{P}_{t|k} + \boldsymbol{P}_{t|k}\boldsymbol{A}(t)^T + \boldsymbol{\sigma}(t)\boldsymbol{\sigma}(t)^T \tag{11.40}$$

with initial conditions given by $\hat{\boldsymbol{x}}_{k|k}$ and $\boldsymbol{P}_{k|k}$.

For the non-linear case

$$d\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t, \boldsymbol{u}_t, t, \boldsymbol{\theta})dt + \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta})d\boldsymbol{\omega}_t \tag{11.41}$$ {eq:ConEKFSys}

$$\boldsymbol{Y}_k = \boldsymbol{h}(\boldsymbol{X}_k, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) + \boldsymbol{e}_k \tag{11.42}$$

we introduce the Jacobian of $\boldsymbol{f}$ around the expectation of $\boldsymbol{X}_t$ ($\hat{\boldsymbol{x}}_t = E[\boldsymbol{X}_t]$), we will use the following short hand notation

$$\boldsymbol{A}(t) = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}_t, t, \boldsymbol{\theta})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{t|k}}, \quad \boldsymbol{f}(t) = \boldsymbol{f}(\hat{\boldsymbol{x}}_{t|k}, \boldsymbol{u}_t, t, \boldsymbol{\theta}) \tag{11.43}$$

where $\hat{\boldsymbol{x}}_t$ is the expectation of $\boldsymbol{X}_t$ at time t, this implies that we can write the first order Taylor expansion of (11.41) as

$$d\boldsymbol{X}_t \approx \left[\boldsymbol{f}(t) + \boldsymbol{A}(t)(\boldsymbol{X}_t - \hat{\boldsymbol{x}}_{t|k})\right] dt + \boldsymbol{\sigma}(t)d\boldsymbol{\omega}_t. \tag{11.44}$$ {eq:ConEKFAppSys}

Using the results from the linear time varying system above we get the following approximate solution to the (11.44)

$$\frac{d\hat{\boldsymbol{x}}_{t|k}}{dt} \approx \boldsymbol{f}(t) \tag{11.45}$$ {eq:EKFEpred}

$$\frac{d\boldsymbol{P}_{t|k}}{dt} \approx \boldsymbol{A}(t)\boldsymbol{P}_{t|k} + \boldsymbol{P}_{t|k}\boldsymbol{A}^T(t) + \boldsymbol{\sigma}(t)\boldsymbol{\sigma}^T(t) \tag{11.46}$$ {eq:EKFVpred}

with initial conditions $E[\boldsymbol{X}_{t_k}] = \hat{\boldsymbol{x}}_{k|k}$ and $V[\boldsymbol{X}_{t_k}] = \boldsymbol{P}_{k|k}$. Equations (11.45) and (11.46) constitute the basis of the prediction step in the Extended Kalman Filter, which for completeness is given below

{the:SDEEKF}

Theorem 2 (Continuous-discrete time Extended Kalman Filter) *With given initial conditions for the $\hat{\boldsymbol{x}}_{1|0} = \boldsymbol{x}_0$ and $\boldsymbol{P}_{1|0} = \boldsymbol{V}_0$ the Extended Kalman Filter approximations are given by; the output* prediction *equations:*

$$\hat{\boldsymbol{y}}_{k|k-1} = \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \tag{11.47}$$ {eqEKFOutPred1}

$$\boldsymbol{R}_{k|k-1} = \boldsymbol{C}_k\boldsymbol{P}_{k|k-1}\boldsymbol{C}_k^T + \boldsymbol{S}_k \tag{11.48}$$ {eqEKFOutPred2}

the innovation *and Kalman* gain *equation:*

$$\boldsymbol{\epsilon}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}; \tag{11.49}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}_k^T\left(\boldsymbol{R}_{k|k-1}\right)^{-1} \tag{11.50}$$

the updating *equations:*

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \boldsymbol{\epsilon}_k; \tag{11.51}$$ {eqEKFUpdate1}

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k \boldsymbol{R}_{k|k-1} \boldsymbol{K}_k^T \tag{11.52}$$ {eqEKFUpdate2}

and the state prediction *equations:*

$$\frac{d\hat{\boldsymbol{x}}_{t|k}}{dt} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{t|k}, \boldsymbol{u}_t, t, \boldsymbol{\theta}) \, , \, t \in [t_k, t_{k+1}[\tag{11.53}$$ {eqEKFPred1}

$$\frac{d\boldsymbol{P}_{t|t_k}}{dt} = \boldsymbol{A}(t)\boldsymbol{P}_{t|t_k} + \boldsymbol{P}_{t|t_k}\boldsymbol{A}(t)^T + \boldsymbol{\sigma}(t)\boldsymbol{\sigma}(t)^T \, , \, t \in [t_k, t_{k+1}[\tag{11.54}$$ {eqEKFPred2}

where the following shorthand notation has been applied:

$$\boldsymbol{A}(t) = \left. \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}_t, t, \boldsymbol{\theta})}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{t|k-1}} \qquad \boldsymbol{C}_k = \left. \frac{\partial \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}_{t_k}, t_k, \boldsymbol{\theta})}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{k|k-1}} \tag{11.55}$$

$$\boldsymbol{\sigma}(t) = \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta}) \qquad \boldsymbol{S}_k = \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \tag{11.56}$$

The ODEs are solved by numerical integration schemes[1], which ensures intelligent re-evaluation of $\boldsymbol{A}$ and $\boldsymbol{\sigma}$ in (11.54).

The prediction step was covered above and the updating step can be derived from linearization of the observation equation and the projection theorem ([**Jazwinski70**]). From the construction above it is clear that the approximation is only likely to hold if the nonlinearities are not too strong. This implies that the sampling frequency is fast enough for the prediction equations to be a good approximation and that the accuracy in the observation equation is good enough for the Gaussian approximation to hold approximately. Even though "simulation of mean", through the prediction equation, is available in CTSM-R, it is recommended that mean simulation results are verified (or indeed performed), by real stochastic simulations (e.g. by simple Euler simulations).

11.3 Iterated extended Kalman filtering

The sensitivity of the extended Kalman filter to nonlinear effects not only means that the approximation to the true state propagation solution provided by the solution to the state prediction equations (11.53) and (11.54) may be too crude. The presence of such effects in the output prediction equations (11.47) and (11.48) may also influence the performance of the filter. An option has therefore been included in CTSM-R for applying the *iterated extended Kalman filter* [**Jazwinski70**], which is an iterative version of the extended Kalman filter that consists of the modified output prediction equations:

$$\hat{\boldsymbol{y}}^i_{k|k-1} = \boldsymbol{h}(\eta_i, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \tag{11.57}$$

$$\boldsymbol{R}^i_{k|k-1} = \boldsymbol{C}_i \boldsymbol{P}_{k|k-1} \boldsymbol{C}_i^T + \boldsymbol{S} \tag{11.58}$$

[1]The specific implementation is based on the algorithms of [**Hindmarsh83**], and to be able to use this method to solve (11.53) and (11.54) simultaneously, the n-vector differential equation in (11.53) has been augmented with an $n \times (n+1)/2$-vector differential equation corresponding to the symmetric $n \times n$-matrix differential equation in (11.54).

the modified innovation equation:

$$\boldsymbol{\epsilon}_k^i = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}^i \tag{11.59}$$

the modified Kalman gain equation:

$$\boldsymbol{K}_k^i = \boldsymbol{P}_{k|k-1}\boldsymbol{C}_i^T(\boldsymbol{R}_{k|k-1}^i)^{-1} \tag{11.60}$$

and the modified updating equations:

$$\eta_{i+1} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{\epsilon}_k^i - \boldsymbol{C}_i(\hat{\boldsymbol{x}}_{k|k-1} - \eta_i)) \tag{11.61}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k^i\boldsymbol{R}_{k|k-1}^i(\boldsymbol{K}_k^i)^T \tag{11.62}$$

where:

$$\boldsymbol{C}_i = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_t}\right|_{\boldsymbol{x}=\eta_i, \boldsymbol{u}=\boldsymbol{u}_k, t=t_k, \boldsymbol{\theta}} \tag{11.63}$$

and $\eta_1 = \hat{\boldsymbol{x}}_{k|k-1}$. The above equations are iterated for $i = 1, \ldots, M$, where M is the maximum number of iterations, or until there is no significant difference between consecutive iterates, whereupon $\hat{\boldsymbol{x}}_{k|k} = \eta_M$ is assigned. This way, the influence of nonlinear effects in (11.47) and (11.48) can be reduced.

12 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:

$$p(\boldsymbol{\theta}|\mathcal{Y}_N) = \frac{p(\mathcal{Y}_N|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{Y}_N)} \propto p(\mathcal{Y}_N|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{12.1}$$

and subsequently finding the parameters that maximize this function, i.e. by performing *maximum a posteriori* (MAP) estimation. A nice feature of this expression is the fact that it reduces to the likelihood function, when no prior information is available ($p(\boldsymbol{\theta})$ uniform), making ML estimation a special case of MAP estimation. In fact, this formulation also allows MAP estimation on a subset of the parameters ($p(\boldsymbol{\theta})$ partly uniform). By introducing the notation[1]:

$$\boldsymbol{\mu}_{\boldsymbol{\theta}} = E\{\boldsymbol{\theta}\} \tag{12.2}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = V\{\boldsymbol{\theta}\} \tag{12.3}$$

and:

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}} = \boldsymbol{\theta} - \boldsymbol{\mu}_{\boldsymbol{\theta}} \tag{12.4}$$

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

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{Y}_N) \propto & \left(\prod_{k=1}^{N} \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1} \boldsymbol{\epsilon}_k\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1})}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0|\boldsymbol{\theta}) \\ & \times \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\right)}{\sqrt{\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})}\left(\sqrt{2\pi}\right)^p} \end{aligned} \tag{12.5}$$

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

$$\begin{aligned} -\ln\left(p(\boldsymbol{\theta}|\mathcal{Y}_N, \boldsymbol{y}_0)\right) \propto & \frac{1}{2}\sum_{k=1}^{N}\left(\ln(\det(\boldsymbol{R}_{k|k-1})) + \boldsymbol{\epsilon}_k^T \boldsymbol{R}_{k|k-1}^{-1} \boldsymbol{\epsilon}_k\right) \\ & + \frac{1}{2}\left(\left(\sum_{k=1}^{N} l\right) + p\right)\ln(2\pi) \\ & + \frac{1}{2}\ln(\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})) + \frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}} \end{aligned} \tag{12.6}$$

[1] In practice $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is specified as $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \boldsymbol{\sigma}_{\boldsymbol{\theta}} \boldsymbol{R}_{\boldsymbol{\theta}} \boldsymbol{\sigma}_{\boldsymbol{\theta}}$, where $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$ is a diagonal matrix of the prior standard deviations and $\boldsymbol{R}_{\boldsymbol{\theta}}$ is the corresponding prior correlation matrix.

and MAP estimates of the parameters (and optionally of the initial states) can now be determined by solving the following nonlinear optimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \{-\ln\left(p(\boldsymbol{\theta}|\mathcal{Y}_N, \boldsymbol{y}_0)\right)\} \tag{12.7}$$

13 Using multiple independent data sets

If, multiple consecutive, sequences of measurements, i.e. $\mathcal{Y}_{N_1}^1$, $\mathcal{Y}_{N_2}^2$, …, $\mathcal{Y}_{N_i}^i$, …, $\mathcal{Y}_{N_S}^S$, are available. Then a similar estimation method can be applied by expanding the expression for the posterior probability density function to the general form:

$$p(\boldsymbol{\theta}|\mathbf{Y}) \propto \left(\prod_{i=1}^{S} \left(\prod_{k=1}^{N_i} \frac{\exp\left(-\frac{1}{2}(\boldsymbol{\epsilon}_k^i)^T (\boldsymbol{R}_{k|k-1}^i)^{-1} \boldsymbol{\epsilon}_k^i\right)}{\sqrt{\det(\boldsymbol{R}_{k|k-1}^i)}\left(\sqrt{2\pi}\right)^l} \right) p(\boldsymbol{y}_0^i|\boldsymbol{\theta}) \right) \times \frac{\exp\left(-\frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\right)}{\sqrt{\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})}\left(\sqrt{2\pi}\right)^p} \tag{13.1}$$

where:

$$\mathbf{Y} = [\mathcal{Y}_{N_1}^1, \mathcal{Y}_{N_2}^2, \ldots, \mathcal{Y}_{N_i}^i, \ldots, \mathcal{Y}_{N_S}^S] \tag{13.2}$$

and where the individual sequences of measurements are assumed to be stochastically independent. This formulation allows MAP estimation on multiple data sets, but, as special cases, it also allows ML estimation on multiple data sets ($p(\boldsymbol{\theta})$ uniform), MAP estimation on a single data set ($S = 1$) and ML estimation on a single data set ($p(\boldsymbol{\theta})$ uniform, $S = 1$). Further conditioning on:

$$\mathbf{y_0} = [\boldsymbol{y}_0^1, \boldsymbol{y}_0^2, \ldots, \boldsymbol{y}_0^i, \ldots, \boldsymbol{y}_0^S] \tag{13.3}$$

and taking the negative logarithm gives:

{eqSeveralCrit}

$$\begin{aligned} -\ln\left(p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{y_0})\right) \propto & \frac{1}{2} \sum_{i=1}^{S} \sum_{k=1}^{N_i} \left(\ln(\det(\boldsymbol{R}_{k|k-1}^i)) + (\boldsymbol{\epsilon}_k^i)^T (\boldsymbol{R}_{k|k-1}^i)^{-1} \boldsymbol{\epsilon}_k^i \right) \\ & + \frac{1}{2} \left(\left(\sum_{i=1}^{S} \sum_{k=1}^{N_i} l \right) + p \right) \ln(2\pi) \\ & + \frac{1}{2} \ln(\det(\boldsymbol{\Sigma}_{\boldsymbol{\theta}})) + \frac{1}{2} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\epsilon}_{\boldsymbol{\theta}} \end{aligned} \tag{13.4}$$

and estimates of the parameters (and optionally of the initial states) can now be determined by solving the following nonlinear optimisation problem:

{eqSeveralProb}

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \left\{ -\ln\left(p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{y_0})\right) \right\} \tag{13.5}$$

Currently initial values for all data sets have to be equal in order to use multiple datasets in CTSM-R directly, however it is not difficult to program the objective funtion allowing multiple initial values, using predict(), the definition of the log-likelihood, and some optimiser from R.

14 Missing observations

The algorithms of the parameter estimation methods described above also make it easy to handle missing observations, i.e. to account for missing values in the output vector $\boldsymbol{y}_k^i$, for some i and some k, when calculating the terms:

{eqKthTerm1}
$$\frac{1}{2}\sum_{i=1}^{S}\sum_{k=1}^{N_i}\left(\ln(\det(\boldsymbol{R}_{k|k-1}^i)) + (\boldsymbol{\epsilon}_k^i)^T(\boldsymbol{R}_{k|k-1}^i)^{-1}\boldsymbol{\epsilon}_k^i\right) \tag{14.1}$$

and:

{eqKthTerm2}
$$\frac{1}{2}\left(\left(\sum_{i=1}^{S}\sum_{k=1}^{N_i} l\right) + p\right)\ln(2\pi) \tag{14.2}$$

in (13.4). To illustrate this, the case of extended Kalman filtering for NL models is considered, but similar arguments apply in the case of Kalman filtering for LTI and LTV models. The usual way to account for missing or non-informative values in the extended Kalman filter is to formally set the corresponding elements of the measurement error covariance matrix $\boldsymbol{S}$ in (11.48) to infinity, which in turn gives zeroes in the corresponding elements of the inverted output covariance matrix $\boldsymbol{R}_{k|k-1}^{-1}$ and the Kalman gain matrix $\boldsymbol{K}_k$, meaning that no updating will take place in (11.51) and (11.52) corresponding to the missing values. This approach cannot be used when calculating (14.1) and (14.2), however, because a solution is needed which modifies both $\boldsymbol{\epsilon}_k^i$, $\boldsymbol{R}_{k|k-1}^i$ and l to reflect that the effective dimension of $\boldsymbol{y}_k^i$ is reduced. This is accomplished by replacing (1.2) with the alternative measurement equation:

$$\overline{\boldsymbol{y}}_k = \boldsymbol{E}\left(\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) + \boldsymbol{e}_k\right) \tag{14.3}$$

where $\boldsymbol{E}$ is an appropriate permutation matrix, which can be constructed from a unit matrix by eliminating the rows that correspond to the missing values in $\boldsymbol{y}_k$. If, for example, $\boldsymbol{y}_k$ has three elements, and the one in the middle is missing. Then the appropriate permutation matrix is given as follows:

$$\boldsymbol{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14.4}$$

Equivalently, the equations of the extended Kalman filter are replaced with the following alternative output prediction equations:

$$\hat{\overline{\boldsymbol{y}}}_{k|k-1} = \boldsymbol{E}\boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \tag{14.5}$$

$$\overline{\boldsymbol{R}}_{k|k-1} = \boldsymbol{E}\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{E}^T + \boldsymbol{E}\boldsymbol{S}\boldsymbol{E}^T \tag{14.6}$$

the alternative innovation equation:

$$\overline{\boldsymbol{\epsilon}}_k = \overline{\boldsymbol{y}}_k - \hat{\overline{\boldsymbol{y}}}_{k|k-1} \tag{14.7}$$

the alternative Kalman gain equation:

$$\overline{\boldsymbol{K}}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{E}^T\overline{\boldsymbol{R}}_{k|k-1}^{-1} \tag{14.8}$$

and the alternative updating equations:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \overline{\boldsymbol{K}}_k\overline{\boldsymbol{\epsilon}}_k \tag{14.9}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \overline{\boldsymbol{K}}_k\overline{\boldsymbol{R}}_{k|k-1}\overline{\boldsymbol{K}}_k^T \tag{14.10}$$

The state prediction equations remain the same, and the above replacements in turn provide the necessary modifications of (14.1) to:

$$\frac{1}{2}\sum_{i=1}^{S}\sum_{k=1}^{N_i}\left(\ln(\det(\overline{\boldsymbol{R}}_{k|k-1}^i)) + (\overline{\boldsymbol{\epsilon}}_k^i)^T(\overline{\boldsymbol{R}}_{k|k-1}^i)^{-1}\overline{\boldsymbol{\epsilon}}_k^i\right) \tag{14.11}$$

whereas modifying (14.2) amounts to a simple reduction of l for the particular values of i and k with the number of missing values in $\boldsymbol{y}_k^i$.

15 Robust Estimation

15.1 Huber's M

The objective function (13.4) of the general formulation (13.5) is quadratic in the innovations $\boldsymbol{\epsilon}_k^i$, and this means that the corresponding parameter estimates are heavily influenced by occasional outliers in the data sets used for the estimation. To deal with this problem, a robust estimation method is applied, where the objective function is modified by replacing the quadratic term:

$$\nu_k^i = (\boldsymbol{\epsilon}_k^i)^T (\boldsymbol{R}_{k|k-1}^i)^{-1} \boldsymbol{\epsilon}_k^i \tag{15.1}$$

with a threshold function $\varphi(\nu_k^i)$, which returns the argument for small values of ν_k^i, but is a linear function of $\boldsymbol{\epsilon}_k^i$ for large values of ν_k^i, i.e.:

$$\varphi(\nu_k^i) = \begin{cases} \nu_k^i & , \ \nu_k^i < c^2 \\ c(2\sqrt{\nu_k^i} - c) & , \ \nu_k^i \geq c^2 \end{cases} \tag{15.2}$$

where $c > 0$ is a constant. The derivative of this function with respect to $\boldsymbol{\epsilon}_k^i$ is known as *Huber's ψ-function* [**Huber81**] and belongs to a class of functions called influence functions, because they measure the influence of $\boldsymbol{\epsilon}_k^i$ on the objective function. Several such functions are available, but Huber's ψ-function has been found to be most appropriate in terms of providing robustness against outliers without rendering optimisation of the objective function infeasible.

16 Various statistics

{secCTSMTestStat}

Within CTSM-R an estimate of the uncertainty of the parameter estimates is obtained by using the fact that by the central limit theorem the estimator in (13.5) is asymptotically Gaussian with mean $\boldsymbol{\theta}$ and covariance:

$$\boldsymbol{\Sigma}_{\hat{\boldsymbol{\theta}}} = \boldsymbol{H}^{-1} \tag{16.1}$$

where the matrix $\boldsymbol{H}$ is given by:

$$\{h_{ij}\} = -E\left\{\frac{\partial^2}{\partial\theta_i\partial\theta_j}\ln\left(p(\boldsymbol{\theta}|\mathbf{Y},\mathbf{y_0})\right)\right\}, i,j = 1,\ldots,p \tag{16.2}$$

and where an approximation to $\boldsymbol{H}$ can be obtained from:

$$\{h_{ij}\} \approx -\left(\frac{\partial^2}{\partial\theta_i\partial\theta_j}\ln\left(p(\boldsymbol{\theta}|\mathbf{Y},\mathbf{y_0})\right)\right)\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, i,j = 1,\ldots,p \tag{16.3}$$

which is the Hessian evaluated at the minimum of the objective function, i.e. $\boldsymbol{H}|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$. As an overall measure of the uncertainty of the parameter estimates, the negative logarithm of the determinant of the Hessian is computed, i.e.:

$$-\ln\left(\det\left(\boldsymbol{H}_i|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}\right)\right). \tag{16.4}$$

The lower the value of this statistic, the lower the overall uncertainty of the parameter estimates. A measure of the uncertainty of the individual parameter estimates is obtained by decomposing the covariance matrix as follows:

$$\boldsymbol{\Sigma}_{\hat{\boldsymbol{\theta}}} = \boldsymbol{\sigma}_{\hat{\boldsymbol{\theta}}}\boldsymbol{R}\boldsymbol{\sigma}_{\hat{\boldsymbol{\theta}}} \tag{16.5}$$

into $\boldsymbol{\sigma}_{\hat{\boldsymbol{\theta}}}$, which is a diagonal matrix of the standard deviations of the parameter estimates, and $\boldsymbol{R}$, which is the corresponding correlation matrix.

The asymptotic Gaussianity of the estimator in (13.5) also allows marginal t-tests to be performed to test the hypothesis:

$$H_0\text{:}\quad \theta_j = 0 \tag{16.6}$$

against the corresponding alternative:

$$H_1\text{:}\quad \theta_j \neq 0 \tag{16.7}$$

i.e. to test whether a given parameter θ_j is marginally insignificant or not. The test quantity is the value of the parameter estimate divided by the standard deviation of the estimate, and under H_0 this quantity is asymptotically

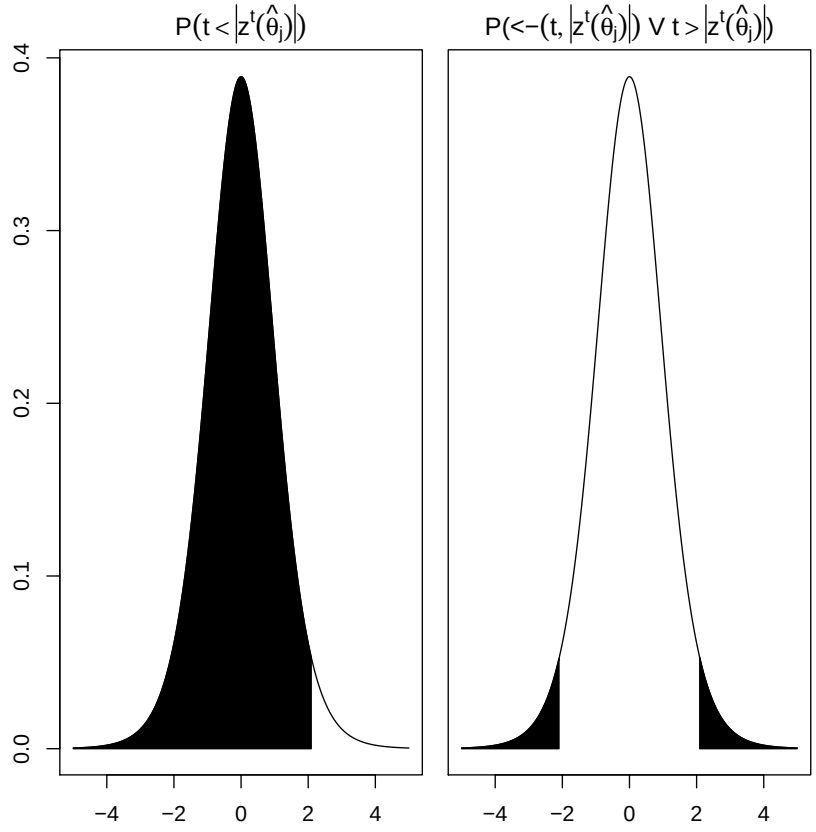

Figure 16.1: Illustration of computation of $P(t<-|z^t(\hat{\theta}_j)| \wedge t>|z^t(\hat{\theta}_j)|)$ via (16.11).

{figProbs}

t-distributed with a number of degrees of freedom (DF) that equals the total number of observations minus the number of estimated parameters, i.e.:

$$z^t(\hat{\theta}_j) = \frac{\hat{\theta}_j}{\sigma_{\hat{\theta}_j}} \quad \in \quad t(\text{DF}) = t\left(\left(\sum_{i=1}^{S}\sum_{k=1}^{N_i} l\right) - p\right) \tag{16.8}$$

where, if there are missing observations in $\boldsymbol{y}_k^i$ for some i and some k, the particular value of log-likelihood (l) is reduced with the number of missing values in $\boldsymbol{y}_k^i$. The critical region for a test on significance level α is given as follows:

$$z^t(\hat{\theta}_j) < t(\text{DF})_{\frac{\alpha}{2}} \ \vee \ z^t(\hat{\theta}_j) > t(\text{DF})_{1-\frac{\alpha}{2}} \tag{16.9}$$

and to facilitate these tests, CTSM-R computes $z^t(\hat{\theta}_j)$ as well as the probabilities:

$$P\left(t<-|z^t(\hat{\theta}_j)| \vee t>|z^t(\hat{\theta}_j)|\right) \tag{16.10}$$

for $j = 1, \ldots, p$. Figure 16.1 shows how these probabilities should be interpreted and illustrates their computation via the following relation:

{eqProbCalc}
$$P\left(t < -|z^t(\hat{\theta}_j)| \vee t > |z^t(\hat{\theta}_j)|\right) = 2\left(1 - P(t < |z^t(\hat{\theta}_j)|)\right) \tag{16.11}$$

with $P(t < |z^t(\hat{\theta}_j)|)$ obtained by approximating the cumulative probability density of the t-distribution $t(\mathrm{DF})$ with the cumulative probability density of the standard Gaussian distribution $N(0,1)$ using the test quantity transformation:

$$z^N(\hat{\theta}_j) = z^t(\hat{\theta}_j)\frac{1 - \frac{1}{4\mathrm{DF}}}{\sqrt{1 + \frac{(z^t(\hat{\theta}_j))^2}{2\mathrm{DF}}}} \quad \in \quad N(0,1) \tag{16.12}$$

The cumulative probability density of the standard Gaussian distribution is computed by approximation using a series expansion of the error function.

Validation data generation

{secCTSMValData}

To facilitate e.g. residual analysis, **CTSM** can also be used to generate validation data, i.e. state and output estimates corresponding to a given input data set, using either pure simulation, prediction, filtering or smoothing.

Simulation of mean

The state and output estimates that can be generated by means of pure simulation are $\hat{\boldsymbol{x}}_{k|0}$ and $\hat{\boldsymbol{y}}_{k|0}$, $k = 0, \ldots, N$, along with their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{k|0}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{k|0})}$ and $\mathrm{SD}(\hat{\boldsymbol{y}}_{k|0}) = \sqrt{\mathrm{diag}(\boldsymbol{R}_{k|0})}$, $k = 0, \ldots, N$. The estimates are generated by the (extended) Kalman filter without updating.

Prediction data generation

The state and output estimates that can be generated by prediction are $\hat{\boldsymbol{x}}_{k|k-j}$, $j \geq 1$, and $\hat{\boldsymbol{y}}_{k|k-j}$, $j \geq 1$, $k = 0, \ldots, N$, along with their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{k|k-j}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{k|k-j})}$ and $\mathrm{SD}(\hat{\boldsymbol{y}}_{k|k-j}) = \sqrt{\mathrm{diag}(\boldsymbol{R}_{k|k-j})}$, $k = 0, \ldots, N$. The estimates are generated by the (extended) Kalman filter with updating.

Filtering data generation

The state estimates that can be generated by filtering are $\hat{\boldsymbol{x}}_{k|k}$, $k = 0, \ldots, N$, along with their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{k|k}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{k|k})}$, $k = 0, \ldots, N$. The estimates are generated by the (extended) Kalman filter with updating.

Smoothing data generation

The state estimates that can be generated by smoothing are $\hat{\boldsymbol{x}}_{k|N}$, $k = 0, \ldots, N$, along with their standard deviations $\mathrm{SD}(\hat{\boldsymbol{x}}_{k|N}) = \sqrt{\mathrm{diag}(\boldsymbol{P}_{k|N})}$, $k = 0, \ldots, N$. The estimates are generated by means of a nonlinear smoothing algorithm based on the extended Kalman filter (for a formal derivation of the algorithm, see **[Gelb74]**). The starting point is the following set of formulas:

$$\hat{\boldsymbol{x}}_{k|N} = \boldsymbol{P}_{k|N}\left(\boldsymbol{P}_{k|k-1}^{-1}\hat{\boldsymbol{x}}_{k|k-1} + \overline{\boldsymbol{P}}_{k|k}^{-1}\hat{\bar{\boldsymbol{x}}}_{k|k}\right) \tag{16.13}$$ {eqSmoothMean1}

$$\boldsymbol{P}_{k|N} = \left(\boldsymbol{P}_{k|k-1}^{-1} + \overline{\boldsymbol{P}}_{k|k}^{-1}\right)^{-1} \tag{16.14}$$ {eqSmoothVariance1}

which states that the smoothed estimate can be computed by combining a forward filter estimate based only on past information with a backward filter estimate based only on present and "future" information. The forward filter estimates $\hat{\boldsymbol{x}}_{k|k-1}$, $k = 1, \ldots, N$, and their covariance matrices $\boldsymbol{P}_{k|k-1}$, $k = 1, \ldots, N$, can be computed by means of the EKF formulation given above, which is straightforward. The backward filter estimates $\hat{\overline{\boldsymbol{x}}}_{k|k}$, $k = 1, \ldots, N$, and their covariance matrices $\overline{\boldsymbol{P}}_{k|k}$, $k = 1, \ldots, N$, on the other hand, must be computed using a different set of formulas. In this set of formulas, a transformation of the time variable is used, i.e. $\tau = t_N - t$, which gives the SDE model, on which the backward filter is based, the following system equation:

$$d\boldsymbol{x}_{t_N-\tau} = -\boldsymbol{f}(\boldsymbol{x}_{t_N-\tau}, \boldsymbol{u}_{t_n-\tau}, t_N - \tau, \boldsymbol{\theta})d\tau - \boldsymbol{\sigma}(\boldsymbol{u}_{t_N-\tau}, t_N - \tau, \boldsymbol{\theta})d\boldsymbol{\omega}_\tau \tag{16.15}$$

where $\tau \in [0, t_N]$. The measurement equation remains unchanged. For ease of implementation a coordinate transformation is also introduced, i.e. $\boldsymbol{s}_t = \overline{\boldsymbol{P}}_t^{-1}\hat{\overline{\boldsymbol{x}}}_t$, and the basic set of formulas in (16.13)-(16.14) is rewritten as follows:

{eqSmoothMean2}
$$\hat{\boldsymbol{x}}_{k|N} = \boldsymbol{P}_{k|N}\left(\boldsymbol{P}_{k|k-1}^{-1}\hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{s}_{k|k}\right) \tag{16.16}$$

{eqSmoothVariance2}
$$\boldsymbol{P}_{k|N} = \left(\boldsymbol{P}_{k|k-1}^{-1} + \overline{\boldsymbol{P}}_{k|k}^{-1}\right)^{-1} \tag{16.17}$$

The backward filter consists of the *updating* equations:

$$\boldsymbol{s}_{k|k} = \boldsymbol{s}_{k|k+1} + \boldsymbol{C}_k^T\boldsymbol{S}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) + \boldsymbol{C}_k\hat{\boldsymbol{x}}_{k|k-1}\right) \tag{16.18}$$

$$\overline{\boldsymbol{P}}_{k|k}^{-1} = \overline{\boldsymbol{P}}_{k|k+1}^{-1} + \boldsymbol{C}_k^T\boldsymbol{S}_k^{-1}\boldsymbol{C}_k \tag{16.19}$$

and the *prediction* equations:

$$\frac{d\boldsymbol{s}_{t_N-\tau|k}}{d\tau} = \boldsymbol{A}_\tau^T\boldsymbol{s}_{t_N-\tau|k} - \overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1}\boldsymbol{\sigma}_\tau\boldsymbol{\sigma}_\tau^T\boldsymbol{s}_{t_N-\tau|k} \tag{16.20}$$

$$- \overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1}\left(\boldsymbol{f}(\hat{\boldsymbol{x}}_{t_N-\tau|k}, \boldsymbol{u}_{t_N-\tau}, t_N - \tau, \boldsymbol{\theta}) - \boldsymbol{A}_\tau\hat{\boldsymbol{x}}_{t_N-\tau|k}\right) \tag{16.21}$$

$$\frac{d\overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1}}{d\tau} = \overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1}\boldsymbol{A}_\tau + \boldsymbol{A}_\tau^T\overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1} - \overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1}\boldsymbol{\sigma}_\tau\boldsymbol{\sigma}_\tau^T\overline{\boldsymbol{P}}_{t_N-\tau|k}^{-1} \tag{16.22}$$

which are solved, e.g. by means of an ODE solver, for $\tau \in [\tau_k, \tau_{k+1}]$. In all of the above equations the following simplified notation has been applied:

$$\begin{aligned} \boldsymbol{A}_\tau = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_{t_N-\tau|k}, \boldsymbol{u}_{t_N-\tau}, t_N-\tau, \boldsymbol{\theta}} \;,\; \boldsymbol{C}_k = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}} \\ \boldsymbol{\sigma}_\tau = \boldsymbol{\sigma}(\boldsymbol{u}_{t_N-\tau}, t_N - \tau, \boldsymbol{\theta}) \;,\; \boldsymbol{S}_k = \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \end{aligned} \tag{16.23}$$

Initial conditions for the backward filter are $\boldsymbol{s}_{N|N+1} = \boldsymbol{0}$ and $\overline{\boldsymbol{P}}_{N|N+1}^{-1} = \boldsymbol{0}$, which can be derived from an alternative formulation of (16.16)-(16.17):

$$\hat{\boldsymbol{x}}_{k|N} = \boldsymbol{P}_{k|N}\left(\boldsymbol{P}_{k|k}^{-1}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{s}_{k|k+1}\right) \tag{16.24}$$

$$\boldsymbol{P}_{k|N} = \left(\boldsymbol{P}_{k|k}^{-1} + \overline{\boldsymbol{P}}_{k|k+1}^{-1}\right)^{-1} \tag{16.25}$$

by realizing that the smoothed estimate must coincide with the forward filter estimate for $k = N$. The smoothing feature is only available for NL models.

17 Computational issues

{chapCompIss}

17.1 LTV (taken from linear Kalman filter)

where the following shorthand notation applies in the LTV case:

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{A}(\hat{\boldsymbol{x}}_{t|k-1}, \boldsymbol{u}_t, t, \boldsymbol{\theta}) \, , \boldsymbol{B} = \boldsymbol{B}(\hat{\boldsymbol{x}}_{t|k-1}, \boldsymbol{u}_t, t, \boldsymbol{\theta}) \\ \boldsymbol{C} = \boldsymbol{C}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \, , \boldsymbol{D} = \boldsymbol{D}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{u}_t, t, \boldsymbol{\theta}) \, , \boldsymbol{S} = \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \end{aligned} \tag{17.1}$$

where the following shorthand notation applies in the LTV case:

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{A}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \, , \boldsymbol{B} = \boldsymbol{B}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \\ \boldsymbol{C} = \boldsymbol{C}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \, , \boldsymbol{D} = \boldsymbol{D}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \, , \boldsymbol{S} = \boldsymbol{S}(\boldsymbol{u}_k, t_k, \boldsymbol{\theta}) \end{aligned} \tag{17.2}$$

and the following shorthand notation applies in the LTI case:

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{A}(\boldsymbol{\theta}) \, , \boldsymbol{B} = \boldsymbol{B}(\boldsymbol{\theta}) \\ \boldsymbol{C} = \boldsymbol{C}(\boldsymbol{\theta}) \, , \boldsymbol{D} = \boldsymbol{D}(\boldsymbol{\theta}) \\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{\theta}) \, , \boldsymbol{S} = \boldsymbol{S}(\boldsymbol{\theta}) \end{aligned} \tag{17.3}$$

In order to be able to use (11.27) and (11.28), the integrals of both equations must be computed. For this purpose the equations are rewritten to:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k+1|k} &= e^{\boldsymbol{A}(t_{k+1}-t_k)} \hat{\boldsymbol{x}}_{k|k} + \int_{t_k}^{t_{k+1}} e^{\boldsymbol{A}(t_{k+1}-s)} \boldsymbol{B} \boldsymbol{u}_s ds \\ &= e^{\boldsymbol{A}\tau_s} \hat{\boldsymbol{x}}_{k|k} + \int_{t_k}^{t_{k+1}} e^{\boldsymbol{A}(t_{k+1}-s)} \boldsymbol{B} \left(\boldsymbol{\alpha}(s - t_k) + \boldsymbol{u}_k \right) ds \\ &= \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_{k|k} + \int_0^{\tau_s} e^{\boldsymbol{A}s} \boldsymbol{B} \left(\boldsymbol{\alpha}(\tau_s - s) + \boldsymbol{u}_k \right) ds \\ &= \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_{k|k} - \int_0^{\tau_s} e^{\boldsymbol{A}s} s ds \boldsymbol{B} \boldsymbol{\alpha} + \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \left(\boldsymbol{\alpha} \tau_s + \boldsymbol{u}_k \right) \end{aligned} \tag{17.4}$$

{eqStatePred5}

and:

$$\begin{aligned}\boldsymbol{P}_{k+1|k} &= e^{\boldsymbol{A}(t_{k+1}-t_k)}\boldsymbol{P}_{k|k}\left(e^{\boldsymbol{A}(t_{k+1}-t_k)}\right)^T \\ &\quad + \int_{t_k}^{t_{k+1}} e^{\boldsymbol{A}(t_{k+1}-s)}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}(t_{k+1}-s)}\right)^T ds \\ &= e^{\boldsymbol{A}\tau_s}\boldsymbol{P}_{k|k}\left(e^{\boldsymbol{A}\tau_s}\right)^T + \int_0^{\tau_s} e^{\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}s}\right)^T ds \\ &= \boldsymbol{\Phi}_s\boldsymbol{P}_{k|k}\boldsymbol{\Phi}_s^T + \int_0^{\tau_s} e^{\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}s}\right)^T ds\end{aligned} \tag{17.5}$$

{eqStatePred6}

where $\tau_s = t_{k+1} - t_k$ and $\boldsymbol{\Phi}_s = e^{\boldsymbol{A}\tau_s}$, and where:

$$\boldsymbol{\alpha} = \frac{\boldsymbol{u}_{k+1} - \boldsymbol{u}_k}{t_{k+1} - t_k} \tag{17.6}$$

has been introduced to allow assumption of either *zero order hold* ($\boldsymbol{\alpha} = \boldsymbol{0}$) or *first order hold* ($\boldsymbol{\alpha} \neq \boldsymbol{0}$) on the inputs between sampling instants. The matrix exponential $\boldsymbol{\Phi}_s = e^{\boldsymbol{A}\tau_s}$ can be computed by means of a Padé approximation with repeated scaling and squaring [**MolerVanLoan78**]. However, both $\boldsymbol{\Phi}_s$ and the integral in (17.5) can be computed simultaneously through:

$$\exp\left(\begin{bmatrix}-\boldsymbol{A} & \boldsymbol{\sigma}\boldsymbol{\sigma}^T \\ \boldsymbol{0} & \boldsymbol{A}^T\end{bmatrix}\tau_s\right) = \begin{bmatrix}\boldsymbol{H}_1(\tau_s) & \boldsymbol{H}_2(\tau_s) \\ \boldsymbol{0} & \boldsymbol{H}_3(\tau_s)\end{bmatrix} \tag{17.7}$$

by combining submatrices of the result[1] [**VanLoan78**], i.e.:

$$\boldsymbol{\Phi}_s = \boldsymbol{H}_3^T(\tau_s) \tag{17.8}$$

and:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}s}\right)^T ds = \boldsymbol{H}_3^T(\tau_s)\boldsymbol{H}_2(\tau_s) \tag{17.9}$$

Alternatively, this integral can be computed from the Lyapunov equation:

$$\begin{aligned}\boldsymbol{\Phi}_s\boldsymbol{\sigma}\boldsymbol{\sigma}^T\boldsymbol{\Phi}_s^T - \boldsymbol{\sigma}\boldsymbol{\sigma}^T &= \boldsymbol{A}\int_0^{\tau_s} e^{\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}s}\right)^T ds \\ &\quad + \int_0^{\tau_s} e^{\boldsymbol{A}s}\boldsymbol{\sigma}\boldsymbol{\sigma}^T\left(e^{\boldsymbol{A}s}\right)^T ds\boldsymbol{A}^T\end{aligned} \tag{17.10}$$

but this approach has been found to be less feasible. The integrals in (17.4) are not as easy to deal with, especially if $\boldsymbol{A}$ is singular. However, this problem can be solved by introducing the singular value decomposition (SVD) of $\boldsymbol{A}$, i.e. $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, transforming the integrals and subsequently computing these.

The first integral can be transformed as follows:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s}sds = \boldsymbol{U}\int_0^{\tau_s}\boldsymbol{U}^T e^{\boldsymbol{A}s}\boldsymbol{U}sds\boldsymbol{U}^T = \boldsymbol{U}\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s}sds\boldsymbol{U}^T \tag{17.11}$$

[1] Within **CTSM** the specific implementation is based on the algorithms of [**Sidje98**].

and, if $\boldsymbol{A}$ is singular, the matrix $\tilde{\boldsymbol{A}} = \boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{U} = \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U}$ has a special structure:

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{17.12}$$

which allows the integral to be computed as follows:

$$\begin{aligned}
\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} s ds &= \int_0^{\tau_s} \left(\boldsymbol{I}s + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} s^2 + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}^2 \frac{s^3}{2} + \cdots \right) ds \\
&= \int_0^{\tau_s} \left(\boldsymbol{I}s + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} s^2 + \begin{bmatrix} \tilde{\boldsymbol{A}}_1^2 & \tilde{\boldsymbol{A}}_1\tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \frac{s^3}{2} + \cdots \right) ds \\
&= \begin{bmatrix} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}_1 s} s ds & \int_0^{\tau_s} \tilde{\boldsymbol{A}}_1^{-1} \left(e^{\tilde{\boldsymbol{A}}_1 s} - \boldsymbol{I} \right) s \tilde{\boldsymbol{A}}_2 ds \\ \boldsymbol{0} & \boldsymbol{I}\frac{\tau_s^2}{2} \end{bmatrix} \\
&= \begin{bmatrix} \left[\tilde{\boldsymbol{A}}_1^{-1} e^{\tilde{\boldsymbol{A}}_1 s} \left(\boldsymbol{I}s - \tilde{\boldsymbol{A}}_1^{-1} \right) \right]_0^{\tau_s} & \\ \boldsymbol{0} & \\ & \tilde{\boldsymbol{A}}_1^{-1} \left[\tilde{\boldsymbol{A}}_1^{-1} e^{\tilde{\boldsymbol{A}}_1 s} \left(\boldsymbol{I}s - \tilde{\boldsymbol{A}}_1^{-1} \right) - \boldsymbol{I}\frac{s^2}{2} \right]_0^{\tau_s} \tilde{\boldsymbol{A}}_2 \\ & \boldsymbol{I}\frac{\tau_s^2}{2} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{A}}_1^{-1} \left(-\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) + \tilde{\boldsymbol{\Phi}}_s^1 \tau_s \right) \\ \boldsymbol{0} \\ \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{A}}_1^{-1} \left(-\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) + \tilde{\boldsymbol{\Phi}}_s^1 \tau_s \right) - \boldsymbol{I}\frac{\tau_s^2}{2} \right) \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{I}\frac{\tau_s^2}{2} \end{bmatrix}
\end{aligned} \tag{17.13}$$

where $\tilde{\boldsymbol{\Phi}}_s^1$ is the upper left part of the matrix:

$$\tilde{\boldsymbol{\Phi}}_s = \boldsymbol{U}^T \boldsymbol{\Phi}_s \boldsymbol{U} = \begin{bmatrix} \tilde{\boldsymbol{\Phi}}_s^1 & \tilde{\boldsymbol{\Phi}}_s^2 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{17.14}$$

The second integral can be transformed as follows:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} ds = \boldsymbol{U} \int_0^{\tau_s} \boldsymbol{U}^T e^{\boldsymbol{A}s} \boldsymbol{U} ds \boldsymbol{U}^T = \boldsymbol{U} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds \boldsymbol{U}^T \tag{17.15}$$

and can subsequently be computed as follows:

$$\begin{aligned}
\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds &= \int_0^{\tau_s} \left(\boldsymbol{I} + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} s + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}^2 \frac{s^2}{2} + \cdots \right) ds \\
&= \int_0^{\tau_s} \left(\boldsymbol{I} + \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} s + \begin{bmatrix} \tilde{\boldsymbol{A}}_1^2 & \tilde{\boldsymbol{A}}_1\tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \frac{s^2}{2} + \cdots \right) ds \\
&= \begin{bmatrix} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}_1 s} ds & \int_0^{\tau_s} \tilde{\boldsymbol{A}}_1^{-1} \left(e^{\tilde{\boldsymbol{A}}_1 s} - \boldsymbol{I} \right) \tilde{\boldsymbol{A}}_2 ds \\ \boldsymbol{0} & \boldsymbol{I}\tau_s \end{bmatrix} \\
&= \begin{bmatrix} \left[\tilde{\boldsymbol{A}}_1^{-1} e^{\tilde{\boldsymbol{A}}_1 s} \right]_0^{\tau_s} & \tilde{\boldsymbol{A}}_1^{-1} \left[\tilde{\boldsymbol{A}}_1^{-1} e^{\tilde{\boldsymbol{A}}_1 s} - \boldsymbol{I}s \right]_0^{\tau_s} \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{I}\tau_s \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) & \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) - \boldsymbol{I}\tau_s \right) \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{I}\tau_s \end{bmatrix}
\end{aligned} \tag{17.16}$$

Depending on the specific singularity of $\boldsymbol{A}$ (see Section 17.1 for details on how this is determined in CTSM-R) and the particular nature of the inputs, several different cases are possible as shown in the following.

General case: Singular A, first order hold on inputs

In the general case, the Kalman filter prediction can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_j - \boldsymbol{U} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds \boldsymbol{U}^T \boldsymbol{B} \boldsymbol{\alpha} + \boldsymbol{U} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds \boldsymbol{U}^T \boldsymbol{B} \left(\boldsymbol{\alpha} \tau_s + \boldsymbol{u}_j \right) \tag{17.17}$$

with:

$$\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds = \begin{bmatrix} \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) & \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) - \boldsymbol{I} \tau_s \right) \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{I} \tau_s \end{bmatrix} \tag{17.18}$$

and:

$$\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} s ds = \begin{bmatrix} \tilde{\boldsymbol{A}}_1^{-1} \left(-\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) + \tilde{\boldsymbol{\Phi}}_s^1 \tau_s \right) \\ \boldsymbol{0} \\ \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{A}}_1^{-1} \left(-\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) + \tilde{\boldsymbol{\Phi}}_s^1 \tau_s \right) - \boldsymbol{I} \frac{\tau_s^2}{2} \right) \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{I} \frac{\tau_s^2}{2} \end{bmatrix} \tag{17.19}$$

Special case no. 1: Singular A, zero order hold on inputs

The Kalman filter prediction for this special case can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_j + \boldsymbol{U} \int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds \boldsymbol{U}^T \boldsymbol{B} \boldsymbol{u}_j \tag{17.20}$$

with:

$$\int_0^{\tau_s} e^{\tilde{\boldsymbol{A}}s} ds = \begin{bmatrix} \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) & \tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{A}}_1^{-1} \left(\tilde{\boldsymbol{\Phi}}_s^1 - \boldsymbol{I} \right) - \boldsymbol{I} \tau_s \right) \tilde{\boldsymbol{A}}_2 \\ \boldsymbol{0} & \boldsymbol{I} \tau_s \end{bmatrix} \tag{17.21}$$

Special case no. 2: Nonsingular A, first order hold on inputs

The Kalman filter prediction for this special case can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_j - \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \boldsymbol{\alpha} + \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \left(\boldsymbol{\alpha} \tau_s + \boldsymbol{u}_j \right) \tag{17.22}$$

with:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} ds = \boldsymbol{A}^{-1} \left(\boldsymbol{\Phi}_s - \boldsymbol{I} \right) \tag{17.23}$$

and:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} s ds = \boldsymbol{A}^{-1} \left(-\boldsymbol{A}^{-1} \left(\boldsymbol{\Phi}_s - \boldsymbol{I} \right) + \boldsymbol{\Phi}_s \tau_s \right) \tag{17.24}$$

Special case no. 3: Nonsingular A, zero order hold on inputs

The Kalman filter prediction for this special case can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \boldsymbol{\Phi}_s \hat{\boldsymbol{x}}_j + \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \boldsymbol{u}_j \tag{17.25}$$

with:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} ds = \boldsymbol{A}^{-1} \left(\boldsymbol{\Phi}_s - \boldsymbol{I} \right) \tag{17.26}$$

Special case no. 4: Identically zero A, first order hold on inputs

The Kalman filter prediction for this special case can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \hat{\boldsymbol{x}}_j - \int_0^{\tau_s} e^{\boldsymbol{A}s} s ds \boldsymbol{B} \boldsymbol{\alpha} + \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \left(\boldsymbol{\alpha} \tau_s + \boldsymbol{u}_j \right) \tag{17.27}$$

with:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} ds = \boldsymbol{I} \tau_s \tag{17.28}$$

and:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} s ds = \boldsymbol{I} \frac{\tau_s^2}{2} \tag{17.29}$$

Special case no. 5: Identically zero A, zero order hold on inputs

The Kalman filter prediction for this special case can be calculated as follows:

$$\hat{\boldsymbol{x}}_{j+1} = \hat{\boldsymbol{x}}_j + \int_0^{\tau_s} e^{\boldsymbol{A}s} ds \boldsymbol{B} \boldsymbol{u}_j \tag{17.30}$$

with:

$$\int_0^{\tau_s} e^{\boldsymbol{A}s} ds = \boldsymbol{I} \tau_s \tag{17.31}$$

Determination of singularity

{secSingDet}

Computing the singular value decomposition (SVD) of a matrix is a computationally expensive task, which should be avoided if possible. Within **CTSM** the determination of whether or not the $\boldsymbol{A}$ matrix is singular and thus whether or not the SVD should be applied, therefore is not based on the SVD itself, but on an estimate of the reciprocal condition number, i.e.:

$$\hat{\kappa}^{-1} = \frac{1}{|\boldsymbol{A}||\boldsymbol{A}^{-1}|} \tag{17.32}$$

where $|\boldsymbol{A}|$ is the 1-norm of the $\boldsymbol{A}$ matrix and $|\boldsymbol{A}^{-1}|$ is an estimate of the 1-norm of $\boldsymbol{A}^{-1}$. This quantity can be computed much faster than the SVD, and only if its value is below a certain threshold (e.g. 1e-12), the SVD is applied.

Factorization of covariance matrices

{secCholesky}

The (extended) Kalman filter may be numerically unstable in certain situations. The problem arises when some of the covariance matrices, which are known from theory to be symmetric and positive definite, become non-positive definite because of rounding errors. Consequently, careful handling of the covariance equations is needed to stabilize the (extended) Kalman filter. Within **CTSM**, all covariance matrices are therefore replaced with their square root free Cholesky decompositions [**Fletcher74**], i.e.:

{eqCholFact}

$$\boldsymbol{P} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T \tag{17.33}$$

where $\boldsymbol{P}$ is the covariance matrix, $\boldsymbol{L}$ is a unit lower triangular matrix and $\boldsymbol{D}$ is a diagonal matrix with $d_{ii} > 0, \forall i$. Using factorized covariance matrices, all of the covariance equations of the (extended) Kalman filter can be handled by means of the following equation for updating a factorized matrix:

{eqCholUpdate}

$$\tilde{\boldsymbol{P}} = \boldsymbol{P} + \boldsymbol{G}\boldsymbol{D}_g\boldsymbol{G}^T \tag{17.34}$$

where $\tilde{\boldsymbol{P}}$ is known from theory to be both symmetric and positive definite and $\boldsymbol{P}$ is given by (17.33), and where $\boldsymbol{D}_g$ is a diagonal matrix and $\boldsymbol{G}$ is a full matrix. Solving this equation amounts to finding a unit lower triangular matrix $\tilde{\boldsymbol{L}}$ and a diagonal matrix $\tilde{\boldsymbol{D}}$ with $\tilde{d}_{ii} > 0, \forall i$, such that:

$$\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{L}}^T \tag{17.35}$$

and for this purpose a number of different methods are available, e.g. the method described by [**Fletcher74**], which is based on the modified Givens transformation, and the method described by [**Thornton80**], which is based on the modified weighted Gram-Schmidt orthogonalization. Within **CTSM** the specific implementation of the (extended) Kalman filter is based on the latter, and this implementation has been proven to have a high grade of accuracy as well as stability [**Bierman77**].

Using factorized covariance matrices also facilitates easy computation of those parts of the objective function (13.4) that depend on determinants of covariance matrices. This is due to the following identities:

$$\det(\boldsymbol{P}) = \det(\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T) = \det(\boldsymbol{D}) = \prod_i d_{ii} \tag{17.36}$$

Optimization issues

{secCTSMOptIssues}

CTSM uses a *quasi-Newton* method based on the BFGS updating formula and a soft line search algorithm to solve the nonlinear optimization problem (13.5). This method is similar to the one described by [**Dennis83**], except for the fact that the gradient of the objective function is approximated by a set of finite difference derivatives. In analogy with ordinary Newton-Raphson methods for optimization, quasi-Newton methods seek a minimum of a nonlinear objective function $\mathcal{F}(\boldsymbol{\theta})$: $\mathbb{R}^p \rightarrow \mathbb{R}$, i.e.:

$$\min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta}) \tag{17.37}$$

where a minimum of $\mathcal{F}(\boldsymbol{\theta})$ is found when the gradient $\boldsymbol{g}(\boldsymbol{\theta}) = \frac{\partial \mathcal{F}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ satisfies:

$$\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0} \tag{17.38}$$

Both types of methods are based on the Taylor expansion of $\boldsymbol{g}(\boldsymbol{\theta})$ to first order:

$$\boldsymbol{g}(\boldsymbol{\theta}^i + \boldsymbol{\delta}) = \boldsymbol{g}(\boldsymbol{\theta}^i) + \frac{\partial \boldsymbol{g}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^i}\, \boldsymbol{\delta} + o(\boldsymbol{\delta}) \tag{17.39}$$

which by setting $\boldsymbol{g}(\boldsymbol{\theta}^i + \boldsymbol{\delta}) = \mathbf{0}$ and neglecting $o(\boldsymbol{\delta})$ can be rewritten as follows:

$$\boldsymbol{\delta}^i = -\boldsymbol{H}_i^{-1}\boldsymbol{g}(\boldsymbol{\theta}^i) \tag{17.40}$$ {eqNewtonDirection}

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \boldsymbol{\delta}^i \tag{17.41}$$ {eqNewtonUpdate}

i.e. as an iterative algorithm, and this algorithm can be shown to converge to a (possibly local) minimum. The Hessian $\boldsymbol{H}_i$ is defined as follows:

$$\boldsymbol{H}_i = \frac{\partial \boldsymbol{g}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^i} \tag{17.42}$$

but unfortunately neither the Hessian nor the gradient can be computed explicitly for the optimization problem (13.5). As mentioned above, the gradient is therefore approximated by a set of finite difference derivatives, and a secant approximation based on the BFGS updating formula is applied for the Hessian. It is the use of a secant approximation to the Hessian that distinguishes quasi-Newton methods from ordinary Newton-Raphson methods.

Finite difference derivative approximations

Since the gradient $\boldsymbol{g}(\boldsymbol{\theta}^i)$ cannot be computed explicitly, it is approximated by a set of finite difference derivatives. Initially, i.e. as long as $||\boldsymbol{g}(\boldsymbol{\theta})||$ does not become too small during the iterations of the optimization algorithm, *forward difference* approximations are used, i.e.:

$$g_j(\boldsymbol{\theta}^i) \approx \frac{\mathcal{F}(\boldsymbol{\theta}^i + \delta_j \boldsymbol{e}_j) - \mathcal{F}(\boldsymbol{\theta}^i)}{\delta_j}\,, j = 1, \ldots, p \tag{17.43}$$

where $g_j(\boldsymbol{\theta}^i)$ is the j'th component of $\boldsymbol{g}(\boldsymbol{\theta}^i)$ and $\boldsymbol{e}_j$ is the j'th basis vector. The error of this type of approximation is $o(\delta_j)$. Subsequently, i.e. when $||\boldsymbol{g}(\boldsymbol{\theta})||$ becomes small near a minimum of the objective function, *central difference* approximations are used instead, i.e.:

$$g_j(\boldsymbol{\theta}^i) \approx \frac{\mathcal{F}(\boldsymbol{\theta}^i + \delta_j \boldsymbol{e}_j) - \mathcal{F}(\boldsymbol{\theta}^i - \delta_j \boldsymbol{e}_j)}{2\delta_j}\,, j = 1, \ldots, p \tag{17.44}$$

because the error of this type of approximation is only $o(\delta_j^2)$. Unfortunately, central difference approximations require twice as much computation (twice the number of objective function evalutions) as forward difference approximations, so to save computation time forward difference approximations are used initially. The switch from forward differences to central differences is effectuated for $i > 2p$ if the line search algorithm fails to find a better value of $\boldsymbol{\theta}$.

The optimal choice of step length for forward difference approximations is:

$$\delta_j = \eta^{\frac{1}{2}}\theta_j \tag{17.45}$$

whereas for central difference approximations it is:

$$\delta_j = \eta^{\frac{1}{3}}\theta_j \tag{17.46}$$

where η is the relative error of calculating $\mathcal{F}(\boldsymbol{\theta})$ [**Dennis83**].

The BFGS updating formula

Since the Hessian $\boldsymbol{H}_i$ cannot be computed explicitly, a secant approximation is applied. The most effective secant approximation $\boldsymbol{B}_i$ is obtained with the so-called BFGS updating formula [**Dennis83**], i.e.:

{eqBFGS}

$$\boldsymbol{B}_{i+1} = \boldsymbol{B}_i + \frac{\boldsymbol{y}_i\boldsymbol{y}_i^T}{\boldsymbol{y}_i^T\boldsymbol{s}_i} - \frac{\boldsymbol{B}_i\boldsymbol{s}_i\boldsymbol{s}_i^T\boldsymbol{B}_i}{\boldsymbol{s}_i^T\boldsymbol{B}_i\boldsymbol{s}_i} \tag{17.47}$$

where $\boldsymbol{y}_i = \boldsymbol{g}(\boldsymbol{\theta}_{i+1}) - \boldsymbol{g}(\boldsymbol{\theta}_i)$ and $\boldsymbol{s}_i = \boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i$. Necessary and sufficient conditions for $\boldsymbol{B}_{i+1}$ to be positive definite is that $\boldsymbol{B}_i$ is positive definite and that:

{eqBFGSCondition}

$$\boldsymbol{y}_i^T\boldsymbol{s}_i > 0 \tag{17.48}$$

This last demand is automatically met by the line search algorithm. Furthermore, since the Hessian is symmetric and positive definite, it can also be written in terms of its square root free Cholesky factors, i.e.:

$$\boldsymbol{B}_i = \boldsymbol{L}_i\boldsymbol{D}_i\boldsymbol{L}_i^T \tag{17.49}$$

where $\boldsymbol{L}_i$ is a unit lower triangular matrix and $\boldsymbol{D}_i$ is a diagonal matrix with $d_{jj}^i > 0, \forall j$, so, instead of solving (17.47) directly, $\boldsymbol{B}_{i+1}$ can be found by updating the Cholesky factorization of $\boldsymbol{B}_i$ as shown in Section 17.1.

The soft line search algorithm

With $\boldsymbol{\delta}^i$ being the secant direction from (17.40) (using $\boldsymbol{H}_i = \boldsymbol{B}_i$ obtained from (17.47)), the idea of the soft line search algorithm is to replace (17.41) with:

{eqLineSearch}

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \lambda_i\boldsymbol{\delta}^i \tag{17.50}$$

and choose a value of $\lambda_i > 0$ that ensures that the next iterate decreases $\mathcal{F}(\boldsymbol{\theta})$ and that (17.48) is satisfied. Often $\lambda_i = 1$ will satisfy these demands and (17.50) reduces to (17.41). The soft line search algorithm is globally convergent if each step satisfies two simple conditions. The first condition is that the decrease in $\mathcal{F}(\boldsymbol{\theta})$ is sufficient compared to the length of the step $\boldsymbol{s}_i = \lambda_i\boldsymbol{\delta}^i$, i.e.:

{eqLineSearchCond1}

$$\mathcal{F}(\boldsymbol{\theta}^{i+1}) < \mathcal{F}(\boldsymbol{\theta}^i) + \alpha\boldsymbol{g}(\boldsymbol{\theta}^i)^T\boldsymbol{s}_i \tag{17.51}$$

where $\alpha \in]0,1[$. The second condition is that the step is not too short, i.e.:

{eqLineSearchCond2}

$$\boldsymbol{g}(\boldsymbol{\theta}^{i+1})^T\boldsymbol{s}_i \geq \beta\boldsymbol{g}(\boldsymbol{\theta}^i)^T\boldsymbol{s}_i \tag{17.52}$$

where $\beta \in]\alpha,1[$. This last expression and $\boldsymbol{g}(\boldsymbol{\theta}^i)^T\boldsymbol{s}_i < 0$ imply that:

$$\boldsymbol{y}_i^T\boldsymbol{s}_i = \left(\boldsymbol{g}(\boldsymbol{\theta}^{i+1}) - \boldsymbol{g}(\boldsymbol{\theta}^i)\right)^T\boldsymbol{s}_i \geq (\beta-1)\boldsymbol{g}(\boldsymbol{\theta}^i)^T\boldsymbol{s}_i > 0 \tag{17.53}$$

which guarantees that (17.48) is satisfied. The method for finding a value of λ_i that satisfies both (17.51) and (17.52) starts out by trying $\lambda_i = \lambda_p = 1$. If this trial value is not admissible because it fails to satisfy (17.51), a decreased value is found by cubic interpolation using $\mathcal{F}(\boldsymbol{\theta}^i)$, $\boldsymbol{g}(\boldsymbol{\theta}^i)$, $\mathcal{F}(\boldsymbol{\theta}^i + \lambda_p \boldsymbol{\delta}^i)$ and $\boldsymbol{g}(\boldsymbol{\theta}^i + \lambda_p \boldsymbol{\delta}^i)$. If the trial value satisfies (17.51) but not (17.52), an increased value is found by extrapolation. After one or more repetitions, an admissible λ_i is found, because it can be proved that there exists an interval $\lambda_i \in [\lambda_1, \lambda_2]$ where (17.51) and (17.52) are both satisfied [**Dennis83**].

Constraints on parameters

In order to ensure stability in the calculation of the objective function in (13.4), simple constraints on the parameters are introduced, i.e.:

$$\theta_j^{\min} < \theta_j < \theta_j^{\max} \, , j = 1, \ldots, p \tag{17.54}$$

These constraints are satisfied by solving the optimization problem with respect to a transformation of the original parameters, i.e.:

$$\tilde{\theta}_j = \ln \left(\frac{\theta_j - \theta_j^{\min}}{\theta_j^{\max} - \theta_j} \right) \, , j = 1, \ldots, p \tag{17.55}$$

A problem arises with this type of transformation when θ_j is very close to one of the limits, because the finite difference derivative with respect to θ_j may be close to zero, but this problem is solved by adding an appropriate penalty function to (13.4) to give the following modified objective function:

$$\mathcal{F}(\boldsymbol{\theta}) = -\ln \left(p(\boldsymbol{\theta} | \mathbf{Y}, \mathbf{y_0}) \right) + P(\lambda, \boldsymbol{\theta}, \boldsymbol{\theta}^{\min}, \boldsymbol{\theta}^{\max}) \tag{17.56}$$

which is then used instead. The penalty function is given as follows:

$$P(\lambda, \boldsymbol{\theta}, \boldsymbol{\theta}^{\min}, \boldsymbol{\theta}^{\max}) = \lambda \left(\sum_{j=1}^{p} \frac{|\theta_j^{\min}|}{\theta_j - \theta_j^{\min}} + \sum_{j=1}^{p} \frac{|\theta_j^{\max}|}{\theta_j^{\max} - \theta_j} \right) \tag{17.57}$$

for $|\theta_j^{\min}| > 0$ and $|\theta_j^{\max}| > 0$, $j = 1, \ldots, p$. For proper choices of the Lagrange multiplier λ and the limiting values $\theta_j^{\min}$ and $\theta_j^{\max}$ the penalty function has no influence on the estimation when θ_j is well within the limits but will force the finite difference derivative to increase when θ_j is close to one of the limits.

Along with the parameter estimates **CTSM** computes normalized (by multiplication with the estimates) derivatives of $\mathcal{F}(\boldsymbol{\theta})$ and $P(\lambda, \boldsymbol{\theta}, \boldsymbol{\theta}^{\min}, \boldsymbol{\theta}^{\max})$ with respect to the parameters to provide information about the solution. The derivatives of $\mathcal{F}(\boldsymbol{\theta})$ should of course be close to zero, and the absolute values of the derivatives of $P(\lambda, \boldsymbol{\theta}, \boldsymbol{\theta}^{\min}, \boldsymbol{\theta}^{\max})$ should not be large compared to the corresponding absolute values of the derivatives of $\mathcal{F}(\boldsymbol{\theta})$, because this indicates that the corresponding parameters are close to one of their limits.

List of Corrections