Fitting Structural Time Series Models with the R Package stsm

Javier López-de-Lacalle

July 2014
DRAFT: PLEASE DO NOT CITE WITHOUT PERMISSION

Abstract

Structural time series models are a flexible approach for time series analysis. They cover a broad range of models where trend and seasonal components are explicitly modelled. Yet it is not always easy to get reliable parameter estimates by means of the standard procedure. In this paper, we introduce a number of utilities available in R to fit these models. Upon the functionalities available in the packages stsm.class and KFKSDS, the package stsm implements maximum likelihood methods in the time and frequency domains.

1 Introduction

Structural time series models (STSM) are a useful approach for time series analysis. A STSM consists of building blocks defining a submodel for different unobserved or latent components such as trend, cycle or seasonal. The practitioner chooses the structures and the models for those components that are relevant in explaining the dynamics observed in the data. These models are also known as unobserved components models. In order to extract the unobserved components, the Kalman filter is run upon the state space form of the model. That is why these models are sometimes referred to as state space models. However, this name involves a wider class of models that includes a framework alternative to the approach described above. For example, autoregressive moving-average models (ARMA) can be cast into state space form. Despite the ARMA model can be decomposed into a trend, cycle and seasonal components, those components do not appear explicitly in the definition and selection of the ARMA model.

As witnessed in the special issue of the *Journal of Statistical Software* (Commandeur *et al.*, 2011), there are several statistical software packages providing support for structural time series models. The papers in that special issue show the usage and syntax of the software and illustrate some of the more particular functionalities provided by each package such as, multivariate analysis, Bayesian methods or diagnostic tests. The wide variety of software tools and packages reveals the relevance of structural models for time series analysis. However, the real difficulties that practitioners may find when working with STSM remain undisclosed. Indeed, as stated in the documentation of the function `StructTS`, *optimization of structural models is a lot harder than many of the references admit.* Sometimes it is not possible to replicate the results from a

benchmark application. Discrepancies in results may arise due to choices in the design of the algorithm that are not available in a software package or are not reported by the source. Other times, it may happen that, given a design of the model, two algorithms converge to different local optima.

Although state space methods are implemented in several R packages (Tusell, 2011), there are few alternative optimization algorithms to fit structural time series models. The common procedure to fit a STSM is as follows: 1) choose arbitrary starting values for the parameters, 2) evaluate the log-likelihood function by means of the Kalman filter, 3) obtain a new set of parameter values by means of the L-BFGS-B algorithm, 4) iterate the searching procedure until a predetermined degree of convergence. This is the way of proceeding that we find in the function `StructTS` of the R stats core package and in contributed packages related to STSM. As alternative methods, the package sspir implements the Expectation-Maximization algorithm and the package dlm (Petris, 2010), implements Bayesian methods to fit these models. Despite the apparent simplicity of the above procedure, an implementation of the Kalman filter along with a general purpose optimization algorithm is not enough to fit these models to real data. In this paper, we discuss both methodological and practical issues regarding the estimation of STSM that are often ignored. We describe optimization algorithms specific for structural time series models and show their usage with the stsm package.

The remaining of the paper is organized as follows. Section 2 describes the basic structural model. The R packages that are the object of this paper are introduced in Section 3. Algorithms and functions to fit a model by maximum likelihood are covered in Sections 4 and 5, respectively for the time and frequency domains. The traditional and a modified Expectation-Maximization algorithm are introduced is Section 6. An application of a non-pure variance model is shown in Section 7. Section 8 concludes.

2 The basic structural model

The basic structural model (BSM) is a pure variance structural model commonly used in applications. It is a relatively broad model and practitioners often select restricted versions of the model. This model plays a central role in the approach advocated in Harvey (1989) for time series analysis. A detailed view of the features and theoretical properties of this model can be found, for instance, in Harvey (1989, Chapter 2), Brockwell and Davis (1996, Chapter 8) and Durbin and Koopman (2001, §3.2). The model is defined as follows:

$$
\begin{aligned}
\textit{observed series:} \quad & y_t = \mu_t + \gamma_t + \epsilon_t, && \epsilon_t \sim \text{NID}(0,\, \sigma^2_\epsilon);\\
\textit{latent level:} \quad & \mu_t = \mu_{t-1} + \beta_{t-1} + \xi_t, && \xi_t \sim \text{NID}(0,\, \sigma^2_\xi);\\
\textit{latent drift:} \quad & \beta_t = \beta_{t-1} + \zeta_t, && \zeta_t \sim \text{NID}(0,\, \sigma^2_\zeta);\\
\textit{latent seasonal:} \quad & \gamma_t = \textstyle\sum_{j=1}^{s-1} -\gamma_{t-j} + \omega_t, && \omega_t \sim \text{NID}(0,\, \sigma^2_\omega),
\end{aligned}
$$

for $t = s, \ldots, n$, where s is the periodicity of the data.

The BSM encompasses models that are common in applications: the local level model, that consists of a random walk with a deterministic drift β_0 plus a noise component ϵ_t; the local trend model, where the drift follows a random walk. Setting $\sigma^2_\omega = 0$ yields a model with deterministic seasonality. Setting also $\gamma_1 = \ldots = \gamma_{s-1} = 0$ removes the seasonal component and gives the local trend model. Adding the restriction $\sigma^2_\zeta = 0$ yields the local level model.

The state space form of the BSM is given by the following representation:

$$\begin{array}{ll} y_t = Z\alpha_t + \epsilon_t, & \epsilon_t \sim \text{NID}(0, \sigma_\epsilon^2), \\ \alpha_t = T\alpha_{t-1} + R\eta_t, & \eta_t \sim \text{NID}(0, V), \\ \alpha_0 \sim \text{N}(a_0, P_0), & \end{array} \qquad \text{with } V = \begin{bmatrix} \sigma_\xi^2 & 0 & 0 \\ 0 & \sigma_\zeta^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{bmatrix},$$

for $t = 1, \ldots, n$. For $s = 4$, the matrices of this representation are defined as follows:

$$\begin{aligned} y_t &= \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \end{bmatrix} \alpha_t + \epsilon_t, \\ \alpha_t \equiv \begin{bmatrix} \mu_t \\ \beta_t \\ \gamma_t \\ \gamma_{t-1} \\ \gamma_{t-2} \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mu_{t-1} \\ \beta_{t-1} \\ \gamma_{t-1} \\ \gamma_{t-2} \\ \gamma_{t-3} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \xi_t \\ \zeta_t \\ \omega_t \end{bmatrix}. \end{aligned}$$

We assume that a_0 and P_0 are known or chosen beforehand. Given the variance parameters, the Kalman filter and smoother can be applied to extract an estimate of the the latent components (level, trend and seasonal).

3 Bundle of packages

The utilities introduced in this paper are implemented in a modular way through a bundle of packages. Each package is specialized in a particular functionality. The package stsm.class defines an S4 class to represent structural time series models and provides some basic methods to work with it. The package KFKSDS performs Kalman filtering and smoothing in state space models. The package stsm contains the main functions for the practical analysis of time series (optimization algorithms, diagnostics, extraction of components). A brief introduction to these packages follows below.

3.1 Package stsm.class

Note: Package stsm.class has been merged into package stsm version 1.5.

The package stsm.class contains the definition of the S4 class named `stsm` and some basic methods to work with it. The `stsm` class defines the main elements of a structural time series models. It is designed bearing in mind that the information about the parameters of the model is needed interchangeably in two forms. Sometimes we need the information as a vector of parameters (e.g. to be passed to an optimization algorithm) while other times we need the whole matrix representation (e.g. to extract the unobserved components by means of the Kalman filter). The slot `pars` is the main container of the parameters. Two other slots may contain information about the parameters. Parameters to be considered fixed in an optimization algorithm are placed in the slot `nopars`. The slot `cpar` contains an optional scaling parameter for the values in `pars`. In practice, `cpar` would be a variance parameter that is concentrated out of the likelihood function. The parameters are labelled as follows. The variance parameters are labelled correlatively from top to bottom according to the representation given in § 2 above. In the basic structural model the vector of parameters `pars` is labelled as `c("var1", "var2", "var3", "var4")`. Similarly, in the local level plus a seasonal component the names of `pars` are

`c("var1", "var2", "var3")`, referring respectively to $\{\sigma^2_\epsilon, \sigma^2_\xi, \sigma^2_\omega\}$. The matrix representation of the selected model is stored in the slot called `ss`. The matrices in this slot contain character strings with the names of the parameters in those places where they are located. The method `char2numeric` returns the matrix in numerical format.

A model with a local level plus a seasonal component can be initialized as follows:

```
R> library("stsm.class")
R> m <- stsm.model(model = "llm+seas", y = JohnsonJohnson)
```

By default the variance parameters of the model are set equal to one. Although the slots can be directly modified using the `@<-` operator, for those users that are not familiar with the internal design of the class, it is safer to use the setter methods defined in the package. The documentation of the method `set.pars` gives some examples showing how the usage of these methods can avoid errors that otherwise would be unnoticed at the time of making the assignment. We can modify the parameters of the model and show their new values as follows:

```
R> m <- set.pars(m, c("var1" = 2, "var2" = 15, "var3" = 30))
R> get.pars(m)
```

```
var1 var2 var3
   2   15   30
```

We may retrieve the matrices of the state space form:

```
R> ss <- char2numeric(m)
R> ss$T
R> ss$V
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0   -1   -1   -1
[3,]    0    1    0    0
[4,]    0    0    1    0
     [,1] [,2]
[1,]   15    0
[2,]    0   30
```

The elements in the list `ss` are named in accordance with the notation given in § 2 above.

Getter methods are also defined. They are convenient when the parameters of the model are scaled relative to `cpar` or when the model is parameterized in terms of an auxiliar set of parameters. The model may be parameterized in terms of an auxiliar set of parameters, θ, in order to prevent a numerical optimization algorithm from searching outside the feasible region. For example, a common parameterization is $\sigma^2 = \theta^2$ to ensure positive variances. The slot `pars` stores the parameters θ (with respect to which the objective function is maximized) while the method `get.pars` returns the corresponding value of the variance parameters. The method `transPars` defines alternative parameterizations. It can also be chosen when creating the `stsm` object.

```
R> m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
+    pars = c(var1 = 2, var2 = 3, var3 = 4), transPars = "square")
R> m@pars

var1 var2 var3
   2    3    4

R> get.pars(m)

var1 var2 var3
   4    9   16
```

The class contains a slot called `ssd` that stores the sample periodogram and a slot called `sgfc` that stores some constant elements related to the spectral generating function of the model. These slots are filled in when creating the object if the arguments `ssd` and `sgfc` of the function `stsm.model` are set to `TRUE`. Having this elements stored in the model object reduces the number of arguments to be passed to some functions in package stsm and avoids the recalculation of constant terms.

3.2 Package KFKSDS

The package KFKSDS implements a *naive* version of the Kalman filter, i.e., a direct implementation of the recursions that can be found in many textbooks, for instance Harvey (1989, Chapter 3), Pollock (1999, Chapter 9), Durbin and Koopman (2001, Chapter 4). This approach may potentially cause numerical problems. In particular, the covariance matrix of the state vector at period t, P_t, may lose the properties of symmetry and non-negative definiteness. As a safeguard against potential numerical instabilities, a square root filter can be used to compute the matrix P_t. For a review on this issue see, for instance, Durbin and Koopman (2001, § 6.3). In our experience with the package and the simulation experiments that we will see, the direct implementation of the Kalman recursions was not troublesome.

There are several other packages in R that perform Kalman filtering in state space models. A review can be found in Tusell (2011). Each one has its own strengths that make them best suited to a particular context. The package KFKSDS was developed as a tool to conduct the work described in López-de-Lacalle (2013a), López-de-Lacalle (2013b). Within this setting, the package is useful as a development tool as well as for debugging and testing purposes.

The package includes a wrapper for the Kalman filter functions available in the following packages: dse version 2013.3-2 (Gilbert, 2013) dlm version 1.1-3 (Petris, 2010; Petris *et al.*, 2009; Petris and Petrone, 2011) (R Development Core Team, 2013; Petris and Petrone, 2011); FKF version 0.1.2 (Luethi *et al.*, 2012); KFAS 0.9.11 (Helske, 2012); sspir version 0.2.10 (Dethlefsen *et al.*, 2012) and function `StructTS` in package stats version 3.0.1. The use of the original interfaces provided by each package is recommended since they sometimes provide further capabilities or options. Nevertheless, the wrapper function provided in this package is useful, for example, for debugging since it allows running different implementations of the filter through a unified interface. The user can easily run any of them for a given model defined as an object of class `stsm`.

A useful utility of the package is that it computes the analytical derivatives of some of the elements of the Kalman filter and smoother with respect to the parameters of the model. In

particular, the necessary elements to compute the analytical derivatives of the time domain log-likelihood function are returned. This is especially useful when it comes to maximizing the likelihood function.

In some models, the Kalman filter and smoother are expected to converge to a steady state (Harvey, 1989, § 3.3, 3.4). The implementation in the package KFKSDS takes account of this fact. Some optional parameters can be defined in order to assess at each iteration of the filter whether a steady state has been reached. When the steady state is reached, the values from the last iteration are used in the remaining iterations instead of doing all the matrix computations. Thus, the number of matrix operations is reduced substantially.

Some parts of the package use precompiled C code where the matrix operations are handled through the GNU Scientific Library (Galassi *et al.*, 2009).

3.3 Package stsm

The packages introduced above provide a set of utilities to define and work with structural time series models as well as common algorithms to extract the latent components of a given model. In the applied analysis of time series, it is necessary to choose optimal parameter values for the model in order to assess the suitability of the model for the sample data. The package stsm contains the main functions for the statistical analysis of time series by means of structural models.

The main strength of the package is the variety of optimization methods that are available to obtain parameter estimates. As discussed in the introduction of this paper, the common approach to fit a structural model is to evaluate the likelihood function by running the Kalman filter and maximize it by means of the L-BFGS-B algorithm. López-de-Lacalle (2013a) shows that in practice there are some issues that are often omitted that may affect the quality of results.

Other R packages related to state space methods are specialized in the Kalman filter. Some of the packages mentioned before include, among other enhancements, square root filters, diffuse initialization and linearization of a model. However, when it comes to obtaining optimal parameter values, the range of tools and methods available in R is relatively poor. As mentioned in the introduction, optimization of structural models may be hard in practice. An implementation of the Kalman filter plus an interface to a numerical optimization algorithm may not be enough to get satisfactory results.

The package stsm contributes dedicated optimization algorithms beyond the more rudimentary approach implemented in the function `StructTS` of the stats package. In the remaining of this paper we will introduce these algorithms and the usage of the package.

4 Maximum likelihood in the time domain

According to the common approach to fit a structural models, it seems that all that is needed is a interface to the Kalman filter and to a numerical optimization algorithm, e.g. L-BFGS-B. López-de-Lacalle (2013a) shows that, in practice, there are several issues to be taken into account by the practitioner. The package stsm provides a flexible implementation that allows the user to try different options that may be best suited for a particular time series. Here, we

will show the use of the `maxlik.td.optim` function to choose some of those options. We use the typical example of the airline passengers data.

First, we replicate the results obtained with the `StructTS` function in the `stats` package. The parameterization of the model is set as `transPars = "StructTS"`. Thus, the likelihood is maximized with respect to a set of parameters θ in such a way that the variance parameters of the model are $\sigma^2 = \theta \times \text{var(y)}/100$, where var(y) is the variance of the data. What is more peculiar is the covariance matrix P_0 of the initial state vector a_0. The elements in the diagonal take a large value (10000 times the variance of the data) to reflect the uncertainty in a_0. The matrix P_0 is generally taken as a diagonal matrix, however, in `StructTS` the values outside the diagonal take on the same value as those in the diagonal. This can be specified including the element `P0cov = TRUE` in the argument `KF.args`.

```
R> res0 <- StructTS(log(AirPassengers), type = "BSM")$coef[c(4,1:3)]
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
+    transPars = "StructTS")
R> res1 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = TRUE), method = "L-BFGS-B", gr = "numerical")
R> mairp <- set.pars(mairp, pmax(res1$par, .Machine$double.eps))
R> round(get.pars(mairp), 6)
R> all.equal(get.pars(mairp), res0, tol = 1e-05, check.att = FALSE)
```

```
    var1     var2     var3     var4
0.000000 0.000772 0.000000 0.001397
[1] TRUE
```

The structure of P_0, i.e., diagonal or not, cannot be chosen through the arguments passed to `StrucTS`. This option is available in `maxlik.td.optim`. Below, we set `P0cov = FALSE`, defining P_0 as a diagonal matrix, and then we repeat the above example.

```
R> mairp <- set.pars(mairp, c(1,1,1,1))
R> res2 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE), method = "L-BFGS-B", gr = "numerical")
R> mairp <- set.pars(mairp, pmax(res2$par, .Machine$double.eps))
R> round(get.pars(mairp), 6)
```

```
    var1     var2     var3     var4
0.000129 0.000700 0.000000 0.000064
```

We can see that the ratio of variances `var2`/`var4`, i.e., the variance of the level over the seasonal components changes from 0.55 to 10.94. Therefore, this seemingly innocuous detail leads to a different allocation of the variance of the data across the components. Although the documentation of `StructTS` claims that the solution found for the `AirPassengers` data is better than the result reported by other source, we cannot be certain beforehand that it will always be the case with any other data set. Next, we will see how the package `stsm` allows the user to choose other options that are often overlooked or at best underestimated.

As an alternative to the L-BFGS-B optimization procedure, the requirement of positive variances can be ensured by parameterizing the model as $\theta = \sigma^2$ and maximizing the likelihood

for the auxiliar vector of parameters θ by means of the BFGS algorithm. For a given structure of P_0, e.g. `P0cov = FALSE` in the example below, the same result as that stored above in `res2` is obtained.

```
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
+    transPars = "square")
R> res3 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE), method = "BFGS", gr = "numerical")
R> round(res3$par^2, 6)

    var1     var2     var3     var4
0.000128 0.000700 0.000000 0.000064
```

Another parameterization that is sometimes used to ensure positive variances, $\sigma^2 = \exp(\theta)$, does not lead in this case to a reliable solution since no role is given to the variance of the seasonal component.

```
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
+    transPars = "exp")
R> res4 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE), method = "BFGS", gr = "numerical")
R> round(exp(res4$par), 6)

    var1     var2     var3     var4
0.000368 0.000766 0.000000 0.000000
```

Given the results obtained above, it seems sensible to set the variance of the slope component equal to zero. The restriction $\sigma_\zeta^2 = 0$ can be set by defining `var3 = 0` in `nopars`.

```
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
+    transPars = "StructTS", nopars = c(var3 = 0))
R> res5 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = TRUE), method = "L-BFGS-B", gr = "numerical")
R> mairp <- set.pars(mairp, res5$par)
R> round(get.pars(mairp), 6)

    var1     var2     var4
0.000000 0.000772 0.001397
```

No major differences are obtained compared to the unrestricted model whose results where stored in `res1` above. Thus, we may then decide to define a model consisting of a local level plus a seasonal component. Instead of setting the restriction $\sigma_\zeta^2 = 0$, the slope component is not included in the definition of the model and the matrices of the state space are redefined accordingly.

```
R> mairp <- stsm.model(model = "llm+seas", y = log(AirPassengers),
+    transPars = "StructTS")
R> res6 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE), method = "L-BFGS-B", gr = "numerical")
```

```
R> mairp <- set.pars(mairp, res6$par)
R> round(get.pars(mairp), 6)
```

```
    var1     var2     var3
0.000028 0.001028 0.000054
```

The local level plus seasonal model contains three variance parameters and, hence, in this model `var3` is referred to the seasonal component. It is striking to see that applying the same optimization procedure on the restricted basic structural model does not lead to the same solution as in the unrestricted `"llm+seas"` model. In the former case the ratio of the level to seasonal variances is 0.55 while in the latter it is 19.04.

An option that is not exploited in the packages available in R is to concentrate one of the parameters out of the likelihood function. The advantage is that the dimension of the optimization problem is reduced in one parameter. Below, we fit the restricted basic structural concentrating out the parameter σ_ξ^2.

```
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
+    nopars = c(var3 = 0), cpar = c(var2 = 1), transPars = "StructTS")
R> res7 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE), method = "L-BFGS-B", gr = "numerical")
R> mairp <- res7$model
R> round(coef(res7), 6)
```

```
    var1     var2     var4
0.000259 0.000393 0.000086
```

As shown in López-de-Lacalle (2013a),[1] the solution obtained above performs well since no seasonality is found in the differenced data and in the residuals. In addition, the estimated seasonal pattern is homoscedastic. Below, we display the estimated components for this solution and compare it with the results based on `StructTS`.

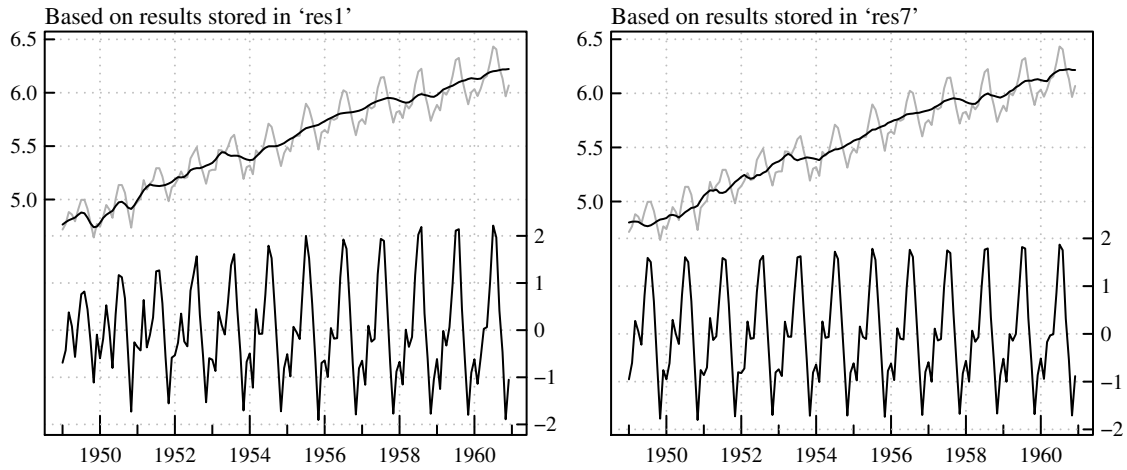
```
R> plot(tsSmooth(StructTS(mairp@y, type = "BSM"))[,1:3])
R> ss <- char2numeric(mairp)
R> kf <- KF(mairp@y, ss)
R> ks <- KS(mairp@y, ss, kf)
R> plot(ks$ahat[,c(1,2,3)])
```

A customized panel of these plots is shown in Figure 1. We can see a relatively homoscedastic component for the seasonal component based on `res7`, while the estimate of the components based on `StructTS` looks shrinked at the beginning of the sample.

Despite we are usually interested in the value of the variance parameters, the likelihood function can also be optimized with respect to the initial state vector, a_0. Brockwell and Davis (1996, § 8.5) proposed a two-steps procedure. The variance parameters are first estimated for a given vector a_0. Then, the vector a_0 is estimated given the variance parameters obtained in

[1] Parameter estimates reported in this source differ slightly with those reported here. The reason is that the expression of the optimal value of the concentrated parameter uses in the denominator the number of observations in the original series while the referenced source used the number of observations in the differenced data. It did not affect the interpretation of the diagnostic tests.

Figure 1: Estimated trend and seasonal components for the logarithms of the airline passengers data. Left hand side plot is based on the procedure implemented in the `StructTS` function, stored in object `res1` above. Right hand side plot is based on the result stored in `res7` above (restricted with $\sigma^2_\zeta = 0$, σ^2_ξ concentrated out of the likelihood and `StructTS` parameterization). The line in gray is the logarithm of the original data.



the previous step. No major differences were found in the diagnostic of the solutions obtained before when the following optimal values of a_0 were considered.

```
R> a0 <- c(mairp@y[1], rep(0, 12))
R> names(a0) <- paste("a0", 1:13, sep = "")
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers), pars = a0,
+    nopars = c("var1" = 0, "var2" = 0.000772, "var3" = 0, "var4" = 0.001397))
R> res8 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = TRUE),
+    method = "L-BFGS-B", gr = "numerical", hessian = FALSE)
R> round(res8$par, 6)
```

```
      a01       a02       a03       a04       a05       a06       a07       a08
 4.794551 -0.005969 -0.121169 -0.236897 -0.090784  0.060561  0.157355  0.162762
      a09       a010      a011      a012      a013
 0.066342 -0.051055  0.012118  0.039345 -0.068661
```

For someone, the variety of solutions that can be found for a given application and model may actually be more illustrative of the nature of numerical optimization than surprising. The reason is that the optimization of the likelihood function by numerical methods does not ensure arriving to a single global optimum. What is clear is the need for further methods and specific algorithms to fit STSM as a complement to general purpose optimization algorithms and interfaces. In what follows, we introduce and show the usage of some of these methods available in the package stsm.

Table 1 reports the performance of some algorithms available in the package stsm. To save space, we do not show here the usage of each function. A similar interface is used by some of the functions introduced in the next section. For details, the user may look at the files `sim-llm-ml-td.R` and `sim-llmseas-ml-td.R` in the `inst` folder of the source package.

These files contain the code that replicates the simulation results shown in Table 1. Since the parameterization of the model is the same in all procedures, we expect results very similar for each procedure. That is what is observed in Table 1 with some exceptions in the local level plus seasonal model. Compared to other procedures, the average parameter estimates for the local level plus seasonal model based on `StructTS` are not close to the true values of the data generating process. The numerical optimization of the likelihood by means of the L-BFGS-B algorithm yields average values very similar to those based on the scoring algorithm, which are in turn very close to the true parameter values. The scoring algorithm is built upon the analytical expressions of the derivatives and the information matrix of the likelihood function. It converges in fewer iterations than the general purpose optimization algorithm L-BFGS-B. Despite the concentration of a parameter reduces the dimension of the optimization problem, the numerical optimization of the concentrated likelihood function did not lead to satisfactory results regardless of the parameter that was chosen to be concentrated, Table 1 reports results when σ^2_ω is concentrated out of the likelihood function. Although not reported, the function `fitSSM` from package KFAS was also used with a diffuse Kalman filter. The performance in terms of parameter estimates and convergence was very similar to that reported for L-BFGS-B. The local level model is a relatively parsimonious model with one component in the state vector. The likelihood surface for this model appears to be rather smooth and all the optimization algorithms reached almost the same solution. The performance in terms of number of iterations required for convergence is still better for the scoring algorithm.

Table 1: Average parameter estimates for simulated data

	Local level model			Local level plus seasonal			
	σ^2_ϵ	σ^2_ξ	#iter.	σ^2_ϵ	σ^2_ξ	σ^2_ω	#iter.
DGP	1600.00	100.00	–	300.00	10.00	100.00	–
`StrucTS`	1609.61	99.09	–	267.89	33.80	131.29	–
L-BFGS-B	1609.33	99.15	34	300.36	9.91	100.38	31
Scoring	1609.02	99.16	9	300.01	9.91	100.57	11
L-BFGS-B con.	1591.88	100.47	–	329.22	9.27	76.64	–

Based on 1000 simulated series. The time domain likelihood function is maximized. DGP: true parameter values used to generate the data. `StructTS` is the function in the `stats` package. L-BFGS-B: `optim` function in package `stats` is used. The scoring algorithm uses analytical expressions of the information matrix. L-BFGS-B con.: maximizes the concentrated likelihood function by means of the L-BFGS-B algorithm. #iter.: average number of iterations until convergence given a tolerance equal to 0.001. Initial parameter values are set equal to 1.

5 Maximum likelihood in the frequency domain

The expression of the log-likelihood function in the frequency domain for a stationary stochastic process with zero mean and positive spectral is given by (Harvey, 1989, § 4.3):

$$\log L(y) = -\frac{n}{2}\log 2\pi - \sum_{j=0}^{[n/2]} \delta_j \log g(\lambda_j) - 2\pi \sum_{j=0}^{[n/2]} \delta_j \frac{I(\lambda_j)}{g(\lambda_j)}\,, \tag{1}$$

where n is the number of observations, $I(\lambda_j)$ is the sample spectral density (periodogram) at frequency λ_j, $g(\lambda_j)$ is the spectral generating model of the the model fitted to the data and δ_j is defined as:

$$\delta_j = \begin{cases} 1/2\,, & \text{if } j = 0, \frac{n}{2} \text{ when } n \text{ is even} \\ 1\,, & \text{elsewhere} \end{cases} .$$

The spectral generating function (s.g.f.) of the basic structural model for a series of periodicity s is defined as:

$$g(\lambda_k) = g_\xi(\lambda_k)\sigma_\xi^2 + g_\zeta(\lambda_k)\sigma_\zeta^2 + g_\omega(\lambda_k)\sigma_\omega^2 + g_\epsilon(\lambda_k)\sigma_\epsilon^2\,,$$

where the following elements are constant, i.e., they do not depend on the parameters of the model:

$$\begin{aligned} g_\xi(\lambda_k) &= 2(1-\cos(\lambda_k s))\,, \quad g_\zeta(\lambda_k) = (1-\cos(\lambda_k s))/(1-\cos(\lambda_k))\,, \\ g_\omega(\lambda_k) &= 4(1-\cos^2(\lambda_k))\,, \quad g_\epsilon(\lambda_k) = 4(1-\cos(\lambda_k))(1-\cos(\lambda_k s))\,. \end{aligned}$$

Maximization of the log-likelihood function in the frequency domain has a number of advantages. The evaluation of the objective function is not an iterative process, no loops are involved on its evaluation. Due to the symmetry of the periodogram and the spectral generating function, vectors of length $[n/2]$ are sufficient to store the information necessary for the calculations. The only element that changes at each iteration of the optimization algorithm is the spectral generating function $g(\lambda_j)$, which in pure variance models contains constant terms as defined above. A potential drawback is that the expression of the likelihood function, equation (1), relies on the assumption that the series is circular. If it is not met, the expression is valid asymptotically and, hence, the value of the likelihood is approximate in finite series.

The package stsm implements the scoring optimization algorithm proposed in Harvey (1989, § 4.3.3). The algorithm is an iterative procedure that starts from an arbitrary set of parameter values and at each iteration chooses an optimal direction and makes a new guess for the parameters. The updating formula at iteration j is given by:

$$\theta^{(j+1)} = \theta^{(j)} + \tau^{(j)} IM\left(\theta^{(j)}\right)^{-1} \frac{\partial \log L}{\partial \theta}\left(\theta^{(j)}\right)\,,$$

where $IM\left(\theta^{(j)}\right)$ is the information matrix evaluated at the set of parameters $\theta^{(j)}$; τ is a positive step size that can be set equal to 1 or chosen at each iteration by means of a line search algorithm. First order derivatives of equation (1) are given by:

$$\frac{\partial \log L}{\partial \theta} = \frac{1}{2}\sum_{j=0}^{T-1}\left(\frac{2\pi I(\lambda_j)}{g(\lambda_j)} - 1\right)\frac{1}{g(\lambda_j)}\frac{\partial g(\lambda_j)}{\partial \theta}\,. \tag{2}$$

In the basic structural model, first order derivatives of the spectral generating function are straightforward to obtain. They are given by the constant terms g_ϵ, g_η, g_ψ and g_ω, respectively for the parameters σ_ϵ^2, σ_η^2, σ_ψ^2 and σ_ω^2. It can be checked that the information matrix with respect to equation (1) is given by:

$$IM(\theta) = \frac{1}{2}\sum_{j=0}^{T-1}\frac{1}{g(\lambda_j)^2}\frac{\partial g(\lambda_j)}{\partial \theta}\frac{\partial g(\lambda_j)}{\partial \theta'}\,. \tag{3}$$

Next, we sketch the implementation of the algorithm described above. We begin by loading a data set containing series generated from the local level plus seasonal model. The data set is

available in the package for testing purposes. Then, we create a `stsm` object defining the model for the first series, `llmseas[,1]`. The initial parameter values are set equal to 1. By setting `ssd` and `sgfc` to `TRUE`, the periodogram and the constant terms of the s.g.f are computed and stored in the model object `m`, in this way they do not need to be recomputed whenever they are required at each iteration of the scoring algorithm. The constant terms are precisely the first order derivatives of the s.g.f and are stored in an object named `sgf.d1` for clarity.

```
R> data("llmseas")
R> m <- stsm.model(model = "llm+seas", y = llmseas[,1],
+    pars = c(var1 = 1, var2 = 1, var3 = 1),
+    ssd = TRUE, sgfc = TRUE)
R> sgf.d1 <- m@sgfc
```

The searching procedure of the scoring algorithm is implemented below. A tolerance equal to 0.001 and a maximum number of iterations equal to 100 are defined to control the convergence of the algorithm. In this sample version of the code, the step size is fixed to 1. The number of iterations are recorded in `iter`.

```
R> convergence <- FALSE
R> tol <- 0.001
R> maxit <- 100
R> step <- 1
R> iter <- 0
R> while (!(convergence || iter > maxit))
+  {
+    sgf <- drop(m@sgfc %*% m@pars)
+    pipgog <- pi * as.vector(m@ssd / sgf)
+    gd1og <- as.matrix(sgf.d1 / sgf)
+    G <- 0.5 * colSums((2 * pipgog - 1) * gd1og)
+    IM <- 0
+    for (i in seq(along = sgf))
+      IM <- IM + 0.5 * tcrossprod(sgf.d1[i,]) / sgf[i]^2
+    pd <- drop(solve(IM) %*% G)
+    pars.old <- m@pars
+    pars.new <- pars.old + step * pd
+    m <- set.pars(m, pars.new)
+    if (sqrt(sum((pars.old - pars.new)^2)) < tol)
+      convergence <- TRUE
+    iter <- iter + 1
+  }
```

The algorithm first computes the spectral generating function, `sgf`, for the current value of the parameters. It is simply the constant terms of the s.g.f multiplied by the parameters. In the version implemented in the package `stsm`, if the parameters defined in the slot `pars` are not stored in the same order as the columns of matrix `sgf`, then the parameters are indexed appropriately. The derivatives given in equation (2) are computed and stored in `G` and the information matrix is computed as defined in equation (3) and stored in `IM`. The optimal searching direction is the

inverse of `IM` times the gradient. It is stored in `pd` and then the parameters and the model object `m` are updated accordingly. Finally, if the Euclidean distance between the current vector of parameters and the vector from the previous iteration is lower than the tolerance `tol`, then `convergence` is set to `TRUE` and the algorithm stops.

Table 2 reports average parameter estimates obtained for 1000 series simulated from the local level and the local level plus seasonal models. Three optimization algorithms are used: the general purpose L-BFGS-B algorithm and two specific algorithms available in the package `stsm`, the Newton-Raphson algorithm (NR) and the scoring algorithm introduced above. The NR algorithm follows an iterative searching procedure in the lines of the scoring algorithm but the analytical Hessian of the likelihood function is used instead of the information matrix. These two algorithms are also applied on a model where the parameter σ_ϵ^2 is concentrated out of the likelihood and, hence, it is not part of the parameters that are searched by the optimization procedure. Both the NR and the scoring algorithms are enhanced with an optimal step size that is chosen at each iteration by means of a line search procedure. In the simulations, the maximum allowed step size is set equal to 1.

Table 2: Average parameter estimates for simulated data

	Local level model			Local level plus seasonal			
	σ_ϵ^2	σ_ξ^2	#iter.	σ_ϵ^2	σ_ξ^2	σ_ω^2	#iter.
DGP	1600.00	100.00	–	300.00	10.00	100.00	–
L-BFGS-B	1596.20	122.97	33	282.70	11.46	112.87	31
NR	1597.15	123.60	23	291.16	11.23	107.79	18
Scoring	1597.15	123.60	8	291.95	11.19	107.73	11
L-BFGS-B con.	1597.00	123.62	6	290.80	11.22	108.39	12
Scoring con.	1597.12	123.61	2	291.40	11.22	107.83	5

Based on 1000 simulated series. The spectral likelihood function is maximized. DGP: true parameter values used to generate the data. L-BFGS-B: `optim` function in package `stats` is used. NR: Newton-Raphson algorithm. In L-BFGS-B con. and Scoring con. the parameter σ_ϵ^2 is concentrated out of the likelihood function. #iter.: average number of iterations until convergence given a tolerance equal to 0.001. Initial parameter values are set equal to 1. See the paths followed by each algorithm for some of the series in the two animated graphics that are provided with this paper.

As expected (since all the elements defining the model are equal), there are no major differences in the result found by each algorithm. Compared to the time domain counterpart reported in Table 1, average parameter estimates are relatively farther from the true values. The reason is that, as mentioned before, the derivation of the spectral likelihood function relies on the assumption of circular data. When it is not met, the expression (1) is approximate. The results in Table 2 reveal that a dedicated optimization algorithm for structural models improves the performance of the procedure. The NR and the scoring algorithms converge to a solution in fewer iterations than the generally applied L-BFGS-B algorithm. The convergence of the algorithms is not reported. We found that the L-BFGS-B algorithm failed to converge in 69 out of the 1000 series simulated from the local level plus seasonal model. The NR converged in all cases, while the scoring algorithm did not converge after 50 iterations in 4 cases. In the

local level model, all the procedures converged to a solution in all series, except for one case where the scoring algorithm did not converge after 50 iterations. Unlike the time domain case -where concentrating a parameter led on average to a worse solution- Table 2 shows that the algorithms perform better when the concentrated likelihood is maximized. Average parameter values are close to the true values, while the number of iterations required for convergence is notable reduced. In the local level plus seasonal model, the L-BFGS-B algorithm applied on the concentrated likelihood converged on average in 12 iterations (compared to 31 in the standard likelihood function); in the scoring algorithm the number of iterations was reduced from 11 to 5 when one parameter is concentrated. As a comprehensive way to view the performance of these algorithms, two animated graphics are provided along with this paper. They display the paths followed by each algorithm from the starting point to the final solution.

The results shown in Table 2 suggests that maximization of the spectral likelihood can be helpful to find initial parameter values. In a second step, these values can be polished by maximization of the computationally more demanding time domain likelihood function.

The step size taken by the NR and scoring algorithms is chosen as the step that maximizes the increase in the likelihood function in the direction where the gradient is projected. This is done by means of a line search procedure that searches the optimum step size within a range of possible values. A line search procedure is usually lightweight to run. In the simulations shown above we considered a maximum step size equal to 1. This is the standard value that is used when the step is fixed for all the iterations and, hence, it is a conservative value. Allowing a higher step size may reduce the number of iterations required for convergence. In the implementation of the package stsm, the choice of the step size plays another important role. The searching interval is bounded within a range that ensures that the variance parameters remain in the region of positive values. Even if a maximum step size equal to 1 is chosen, the algorithm automatically adjusts it to a lower step size if the maximum size is not the optimal or if it leads to negative variances. Despite the parameter values of the data generating processes used in Table 2 are far from zero, we used a starting point where all the parameters are set equal to unity. We observed that, in some cases, at the beginning of the procedure a step size equal to 1 led to negative variances. The choice of the step size as explained above avoided falling outside the feasible region of parameter values. In fact, we also employed the BFGS algorithm implemented in the stats package and found that it led to unreliable results. An insight into it revealed that, as the starting point is relatively close to the bounds, the algorithm did not perform well. A starting point farther from the bounds was required in order to obtain a feasible solution by means of the BFGS algorithm. The choice of the step size as described above is a simple alternative to reparameterizations of the model and to the strategy followed by the L-BFGS-B algorithm to deal with bounds on the parameters.

The scripts that replicate the results of the simulations are available in the folder `inst` of the source files of the stsm package. Below we show how to fit a real series by means of the function `maxlik.fd.scoring`.

```
R> mairp <- stsm.model(model = "BSM", y = log(AirPassengers))
R> res8 <- maxlik.fd.scoring(m = mairp, step = NULL, information = "expected",
+    control = list(maxit = 50, tol = 0.001))
R> round(res8$pars, 6)
```

```
    var1     var2     var3     var4
0.000000 0.001878 0.000637 0.001219
```

The basic structural model is fit to the airline passengers data. Setting `information = "expected"` applies the scoring algorithm. It can be set to `"observed"` in order to run the Newton-Raphson algorithm. If the argument `step` is set equal to a numerical value, that step size in used at all the iterations. If it is `NULL`, the step size is automatically chosen at each iteration of the procedure. Some control parameters (the maximum number of iterations and a tolerance) are passed to determine when the algorithm has converged. Notice that, even in this case where the result is close to the lower bounds of the parameters, a reparameterization of the model or the technique applied by L-BFGS-B were not required in order to ensure positive variance estimates.

Standard errors and confidence intervals

The following estimators of the covariance matrix of the maximum likelihood parameter estimates are available (see Davidson and MacKinnon (2004, § 10.4) for further details):

`hessian:` inverse of the analytical Hessian.

`infomat:` inverse of the analytical expression for the information matrix.

`OPG:` inverse of the outer product of the analytical gradient. This method requires only first order derivatives. It tends to be less reliable in small samples.

`sandwich:` sandwich estimator defined as: $H^{-1}(G'G)H^{-1}$, where G is the gradient vector and H is the Hessian matrix. It requires more computations and may be unreliable in small samples. However, contrary to the previous methods, it is valid when the information matrix equality does not hold due for example to misspecification of the model.

`optimHessian:` inverse of the numerical Hessian returned by `optim`.

Confidence intervals can be computed upon the covariance matrix of the parameter estimates. For example, in the model fitted above for the airline passengers data and stored in `res8`, 95% confidence bands are obtained below. As the Hessian turns out to be not positive definite at this local optimum, the information matrix is used to compute the covariance matrix of the estimates.

```
R> ses <- sqrt(diag(vcov(res8, type = "infomat")))
R> pmax(0, res8$pars - 1.96 * ses)
R> res8$pars + 1.96 * ses
```

It is convenient to use the `confint` method:

```
R> civcov <- confint(res8, type = "vcov", vcov.type = "infomat",
+    level = 0.95)
R> round(civcov, 6)
```

```
Warning message:
In confintStsmFitVcov(object, parm, level, vcov.type) :
  'confint' did not account for the fact that some of the parameters lied
 outside the boundary of the parameter space.
        2.5%    97.5%
```

```
var1 0.000000 0.001648
var2 0.000000 0.004635
var3 0.000114 0.001160
var4 0.000555 0.001882
```

The above approach does not ensure positive lower bounds for the confidence interval. Positive bounds are enforced by chosen the greatest value between zero and the lower bound.

A bootstrap method can also be used to obtain confidence intervals. This approach takes advantage of the following result (Harvey, 1989, § 4.3):

$$\begin{cases} 4\pi I(\lambda_j)/g(\lambda_j) \sim \chi^2_2, & \text{for } j \neq 0, n/2 \text{ (for } n \text{ even)} \\ 2\pi I(\lambda_j)/g(\lambda_j) \sim \chi^2_1, & \text{for } j = 0, n/2 \text{ (for } n \text{ even)} \end{cases}.$$

Upon this result, bootstrap replicates of the periodogram are generated. For each of them parameter estimates are obtained maximizing the spectral likelihood function. The quantiles of the bootstrapped parameter estimates yield the confidence interval. The properties of the frequency domain bootstrap are studied in Dahlhaus and Janas (1996) and has been applied, among others, in Koopman and Wong (2006). An advantage of the bootstrap method is that it yields confidence intervals within the bounds of the parameter, i.e., positive variances. For the same model and data as above we obtain:

```
R> set.seed(123)
R> ciboot <- confint(res8, type = "bootstrap", breps = 100)
R> round(ciboot, 6)
```

```
         2.5%    97.5%
var1 0.000000 0.001241
var2 0.000000 0.002497
var3 0.000407 0.001053
var4 0.000636 0.001550
```

6 Expectation-Maximization algorithm

Despite some practical advantages of the Expectation-Maximization algorithm (EM), its use in the context of structural time series models is limited due to the observed slow convergence. López-de-Lacalle (2013b) proposes a modification that improves the convergence of the algorithm. The original and modified algorithms are available in the package stsm.

Earlier developments of the EM algorithm in the context of structural models were given in Shumway and Stoffer (1982) and Watson and Engle (1983). Koopman and Shephard (1992) and Koopman (1993) develop further calculus on the score vector and the simulation smoother that are involved in the implementation of the EM algorithm. The algorithm is briefly covered in the textbooks Harvey (1989, § 4.2.4), Brockwell and Davis (1996, § 8.7) and Durbin and Koopman (2001, § 7.3.4).

The EM algorithm is characterized by a number of virtues: the likelihood increases at every iteration of the procedure; in some cases neat expressions for the updating equation can be obtained; in certain contexts in time series analysis, it has been found to be robust to poorly chosen starting values of the parameters (Hamilton, 1990); it is self-consistent, which

in addition to other properties implies that the same result is obtained under a variety of changing circumstances (Efron, 1982); it ensures that the result satisfies certain constraints such as non-negative variances. Unfortunately, Watson and Engle (1983) and Harvey and Peters (1990) found that the convergence of the EM algorithm in the context of structural time series models is slow. In particular, as the algorithm approaches the local optimum, the rate of convergence becomes slower. Shumway and Stoffer (1982) noticed this fact as well, pointing it as a disadvantage with respect to other methods.

The modification proposed in López-de-Lacalle (2013b) incorporates information from some derivative terms that are considered null in the original design of the algorithm. The function `maxlik.em` provides an interface to the original and the modified EM algorithm. Below, we apply the original EM algorithm to the first simulated series in the `llmseas` dataset:

```
R> m <- stsm.model(model = "llm+seas", y = llmseas[,1])
R> resem1 <- maxlik.em(model = m, type = "standard",
+    tol = 0.001, maxiter = 250)
R> round(resem1$pars, 6)
R> paste("number of iterations:", resem1$iter)
```

```
      var1       var2       var3
230.629246   6.566464 150.417866
[1] "number of iterations: 250"
```

A similar result is obtained with the modified algorithm, but convergence is achieved in fewer iterations.

```
R> resem2 <- maxlik.em(model = m, type = "modified",
+    tol = 0.001, maxiter = 250)
R> round(resem2$pars, 6)
R> paste("number of iterations:", resem2$iter)
```

```
      var1       var2       var3
230.921830   6.525774 150.300954
[1] "number of iterations: 39"
```

The improvement in convergence is at the cost of computing additional information at each step of the algorithm. The original source (López-de-Lacalle, 2013b) shows detailed results about convergence, timings and other implementations of the same procedure. Mixing the original and the modified algorithms seems an efficient compromise between convergence and speed of the algorithm. Both versions can be mixed using `type = "mix"`. The iterations of the algorithm at which the modified version is run can be specified through the argument `mod.steps`. In the example below, the modified algorithm is run every five iterations (up to the maximum number of allowed iterations, `maxiter = 250`, just in case the algorithm did not converge before) starting from the third iteration.

```
R> resem3 <- maxlik.em(model = m, type = "mix",
+    tol = 0.001, maxiter = 250, mod.steps = seq(3, 250, 5))
R> round(resem3$pars, 6)
R> paste("number of iterations:", resem3$iter)
```

```
      var1       var2       var3
230.899606   6.527821 150.312428
[1] "number of iterations: 75"
```

The mixed EM algorithm returns the following result for the local level plus seasonal model fitted to the `AirPassengers` data.

```
R> mairp <- stsm.model(model = "llm+seas", y = log(AirPassengers))
R> resem.airp <- maxlik.em(mairp, type = "mix", tol = 0.001, maxiter = 250,
+    mod.steps = seq(3, 250, 5))
R> round(resem.airp$pars, 6)
R> paste("number of iterations:", resem.airp$iter)
```

```
    var1     var2     var3
0.001446 0.000863 0.000962
[1] "number of iterations: 15"
```

7 Non pure variance structural model

The general framework of the package stsm is the basic structural model introduced in Section 2. There are many other structural models than can be designed. Some of them may include parameters other than the variances of the unobserved components. For example, Clark (1987) decomposed the real Gross Domestic Product (GDP) into a stochastic trend and a transitory component, where the latter component is modelled as a stationary second order autoregressive process.

$$
\begin{aligned}
y_t &= n_t + x_t\,; \\
n_t &= g_{t-1} + n_{t-1} + v_t, & v_t &\sim \text{NID}(0,\, \sigma_1^2); \\
g_t &= g_{t-1} + w_t, & w_t &\sim \text{NID}(0,\, \sigma_2^2); \\
x_t &= \phi_1 x_{t-1} + \phi_2 x_{t-2} + e_t, & e_t &\sim \text{NID}(0,\, \sigma_3^2),
\end{aligned}
$$

The autoregressive coefficients are subject to more involved constraints in order to keep them inside the region of stationarity. The model defined above can be created with the same interface that we used in the other examples (the function `stsm.model` from package stsm.class). Below, we create a `trend+ar2` model object named `mgdp`. For illustration, we use the same series and the same starting parameter values used in the application shown in Kim and Nelson (1999, § 3.3) that we will take as a reference for illustration.

```
R> data("gdp4795")
R> gdp <- log(gdp4795)
R> pars <- c("var2" = 2, "var3" = 2, "var4" = 2, "phi1" = 2, "phi2" = 1)
R> nopars <- c("var1" = 0, "a01" = 0, "a02" = 0, "a03" = 0, "a04" = 0,
+    "P01" = 100, "P02" = 100, "P03" = 100, "P04" = 100)
R> mgdp <- stsm.model(model = "trend+ar2", y = gdp,
+    pars = pars, nopars = nopars, transPars = "exp10sq")
```

The model is fit following the same procedure as in the above cited reference, i.e. maximization of the time domain likelihood function by means of the BFGS algorithm and using the parameterization labelled `exp10sq` (see documentation of the stsm.class package). A particularity of

the benchmark application is that the first 20 observations are discarded when evaluating the likelihood. Luckily, we can easily adapt our procedure to that case by setting `t0 = 21` in the argument `KF.args` that is passed to the Kalman filter. In this way, the contributions to the likelihood made by the observations from the 21-st to the last one are addep up. The standard deviations of the components are reported instead of the variances.

```
R> res.gdp <- maxlik.td.optim(mgdp, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE, t0 = 21), method = "BFGS")
R> mgdp <- set.pars(mgdp, res.gdp$par)
R> optpars <- c(sqrt(get.pars(mgdp)[1:3]), get.pars(mgdp)[4:5])
R> names(optpars)[1:3] <- paste("stdv", 2:4, sep = "")
R> round(optpars, 6)
```

```
    stdv2     stdv3     stdv4      phi1      phi2
 0.005530  0.000184  0.006174  1.530588 -0.584201
```

As in the benchmark reference, we can display the filtered transitory component as an estimate of the business cycle underlying the real GDP.

```
R> ss <- char2numeric(mgdp)
R> kf <- KF(mgdp@y, ss)
R> plot(window(kf.mgdp$a.upd[,3], start = time(mgdp@y)[21]))
```

Alternatively, we fit a local trend model to the same data:

```
R> mgdp2 <- stsm.model(model = "local-trend", y = gdp, transPars = "square")
R> res2.gdp <- maxlik.td.optim(mgdp2, KF.version = "KFKSDS",
+    KF.args = list(P0cov = FALSE, t0 = 1), method = "BFGS")
R> mgdp2 <- set.pars(mgdp2, get.pars(res2.gdp$model))
```

Figure 2: Cyclical component of real GDP

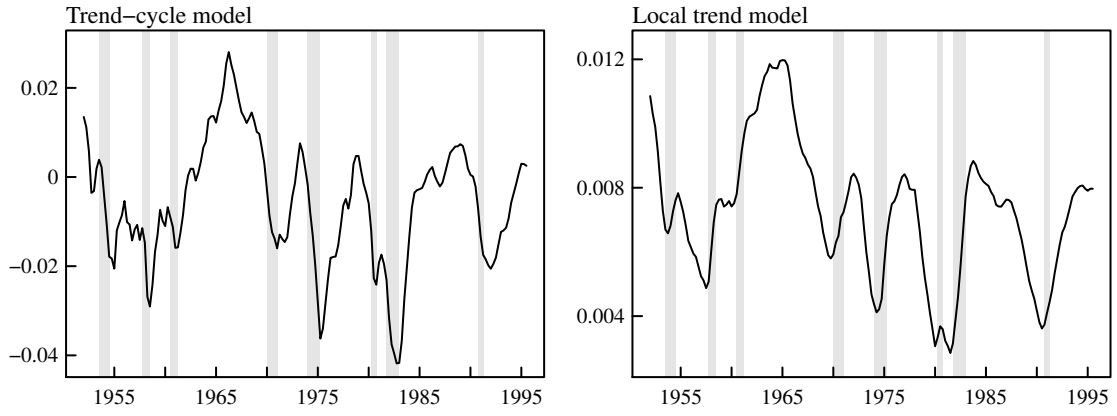


Figure 2 displays the filtered cycle inferred from Clark's model and the smoothed slope component of the local trend model. Gray bars represent the recession periods dated by the NBER.[2] We can see that the changes in the slope of the pure variance local trend model capture

[2] http://www.nber.org/cycles/.

cyclical fluctuations that are similar to those obtained with Clark's model. Both of them closely match the official recession periods.

8 Conclusions

We have illustrated the main features and utilities provided by the bundle of R packages stsm.class, KFKSDS and stsm to fit structural time series models. Upon the utilities available in the first two packages, the main contribution of the stsm package is the implementation of specific algorithms to fit models in the framework of the basic structural time series model. The following enhancements to the standard procedure are implemented: maximization of the time and frequency domain likelihood functions, scoring algorithm based on analytical derivatives, automatic choice of the optimal step size, concentration of a parameter, implementation of the original and a modified version of the expectation-maximization algorithm. We have seen that dedicated optimization algorithms for structural time series models improves the performance of general purpose optimization algorithms that are commonly used.

References

Brockwell PJ, Davis RA (1996). *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer-Verlag.

Clark PK (1987). "The Cyclical Component of U.S. Economic Activity." *Quarterly Journal of Economics*, **102**(4), 797–814.

Commandeur JJF, Koopman SJ, Ooms M (2011). "Statistical Software for State Space Methods." *Journal of Statistical Software*, **41**(1), 1–18. URL http://www.jstatsoft.org/v41/i01/.

Dahlhaus R, Janas D (1996). "A Frequency Domain Bootstrap for Ratio Statistics in Time Series Analysis." *Annals of Statistics*, **24**(5), 1934–1963.

Davidson R, MacKinnon JG (2004). *Econometric Theory and Methods*. Oxford University Press.

Dethlefsen C, Lundbye-Christensen S, Christensen AL (2012). *sspir: State Space Models in R*. R package version 0.2.10, URL http://CRAN.R-project.org/package=sspir.

Durbin J, Koopman SJ (2001). *Time Series Analysis by State Space Methods*. Oxford Statistical Science Series. Oxford University Press.

Efron B (1982). "Maximum Likelihood and Decision Theory." *The Annals of Statistics*, **10**(2), 340–356.

Galassi M, Davies J, Theiler J, Gough B, Jungman G, Booth M, Rossi F (2009). *GNU Scientific Library Reference Manual*. 3rd edition. Network Theory Ltd. ISBN 0954612078.

Gilbert PD (2013). *Brief User's Guide: Dynamic Systems Estimation*. R package version 2013.3-2, URL http://CRAN.R-project.org/package=dse.

Hamilton JD (1990). "Analysis of time series subject to changes in regime." *Journal of Econometrics*, **45**(1–2), 39–70.

Harvey AC (1989). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Harvey AC, Peters S (1990). "Estimation procedures for structural time series models." *Journal of Forecasting*, **9**, 89–108.

Helske J (2012). *KFAS: Kalman Filter and Smoother for Exponential Family State Space Models*. R package version 0.9.11, URL http://CRAN.R-project.org/package=KFAS.

Kim CJ, Nelson CR (1999). *State-Space Models with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications*. The MIT Press.

Koopman SJ (1993). "Disturbance smoother for state space models." *Biometrika*, **80**(1), 117–126.

Koopman SJ, Shephard N (1992). "Exact Score for Time Series Models in State Space Form." *Biometrika*, **79**(4), 823–826.

Koopman SJ, Wong SY (2006). "Extracting Business Cycles using Semi-Parametric Time-varying Spectra with Applications to US Macroeconomic Time Series." *Tinbergen Institute Discussion Papers 2006-105/4*, Tinbergen Institute. URL http://papers.tinbergen.nl/06105.pdf.

Luethi D, Erb P, Otziger S (2012). *FKF: Fast Kalman Filter*. R package version 0.1.2, URL http://CRAN.R-project.org/package=FKF.

López-de-Lacalle J (2013a). "101 Variations on a Maximum Likelihood Procedure for a Structural Time Series Model." Unpublished manuscript.

López-de-Lacalle J (2013b). "Why Does the Expectation-Maximization Algorithm Converge Slowly in Pure Variance Structural Time Series Models?" Unpublished manuscript.

Petris G (2010). *dlm: Bayesian and Likelihood Analysis of Dynamic Linear Models*. R package version 1.1.3, URL http://CRAN.R-project.org/package=dlm.

Petris G, Petrone S (2011). "State Space Models in R." *Journal of Statistical Software*, **41**(4), 1–25. ISSN 1548-7660. URL http://www.jstatsoft.org/v41/i04.

Petris G, Petrone S, Campagnoli P (2009). *Dynamic Linear Models with R*. Use R. Springer. ISBN: 978-0-387-77237-0.

Pollock DSG (1999). *A Handbook of Time-Series Analysis Signal Processing and Dynamics*. Academic Press.

R Development Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Shumway R, Stoffer D (1982). "An approach to time series smoothing and forecasting using the EM algorithm." *Journal of Time Series Analysis*, **3**(4), 253–264.

Tusell F (2011). "Kalman Filtering in R." *Journal of Statistical Software*, **39**(2), 1–27. ISSN 1548-7660. URL http://www.jstatsoft.org/v39/i02/.

Watson MW, Engle RF (1983). "Alternative algorithms for the estimation of dynamic factor, MIMIC and varying coefficient regression models." *Journal of Econometrics*, **23**(3), 385–400.