MS_Regress - The MATLAB Package for Markov Regime Switching Models

Marcelo Perlin*
marceloperlin@gmail.com

First Version: November 23, 2010
This version: April 19, 2015

Abstract

This paper provides an overview of MS_Regress, a MATLAB toolbox specially designed for the estimation, simulation and forecasting of a general markov regime switching model. The package was written in an intuitive manner so that the user have at its reach a large number of different markov switching specifications, without any change in the original code. This document introduces the main functionality of the package with the help of several empirical examples.

*Assistant Professor of Finance, Federal University of Rio Grande do Sul (Porto Alegre, Brazil)

Contents

1 Introduction

The purpose of this document is to introduce the user to the functionality of MS_Regress package[1]. The document is organized as follows, first I give a brief exposition on the topic of regime switching models and the further sections of the paper are related to the introduction to the features of the package along with illustrative examples.

2 Overview of the package

The MS_Regress package was written for the estimation, simulation and forecasting of a general markov switching model. The main functionality of the code is build around three functions:

`MS_Regress_Fit` - Function for estimating a MS model
`MS_Regress_For` - Function for forecasting a regime switching model
`MS_Regress_Sim` - Function for simulating a MS model

Each of these functions have a similar interface. Within the package there are several example scripts that show the functionality of each. In this paper, special attention is given to the fitting function (`MS_Regress_Fit`) since this is the one most likely to the used.

3 Installation of the package

The installation of the package is quite straightforward. The downloaded zip file contains several m files and a few .txt files with the data used in the example scripts. In order to use the main functions, all you need to do is to tell Matlab to place the files from the m_Files folder in the search path:

```
addpath('m_Files');
```

Once `Matlab` recognizes the path of the package, the functions will be

[1]The package is still under development to accommodate new features. The up to date version can be downloaded from `https://sites.google.com/site/marceloperlin/`. I also wrote a lighter version of the package in R. The code is available within the Rmetrics project (`https://r-forge.r-project.org/projects/rmetrics/`, search for fMarkovSwitching). Please be aware that the R version is no longer being maintained.

available to the user. More details about how to use each function can be found in the description of the files[2]. After the use of the package, it is advised (but not necessary) to remove the files from the search path. This is accomplished with:

```
rmpath('m_Files');
```

These commands are already included in all example scripts. These scripts were written so that they can run without any modification. They are a good starting point in learning the interface of the package. Next I give a brief introduction to markov regime switching models.

4 Introduction to markov regime switching models

Markov regime switching models are a type of specification in which the selling point is the flexibility in handling processes driven by heterogeneous states of the world. In this section I give a brief exposition on the subject. Technical details regarding markov regime switching models can be found in Hamilton [1994], Kim and Nelson [1999], Wang [2003]. For introductory material on the subject, see Hamilton [2005], Brooks [2002], Alexander [2008] and Tsay [2002] among others.

Consider the following process given by:

$$y_t = \mu_{S_t} + \epsilon_t \tag{1}$$

where $S_t = 1..k$ and ϵ_t follows a Normal distribution with zero mean and variance given by $\sigma^2_{S_t}$. This is the simplest case of a model with a switching dynamic. Note that for the model given in Equation 1, the intercept is switching states with respect to an indicator variable S_t. This means that if there are k states, there will be k values for μ and σ^2. If there is only one state of the world ($S_t = 1$), formula 1 takes the shape of $y_t = \mu_1 + \epsilon_t$ and it can be treated as a simple linear regression model under general conditions.

Assuming now that the model in 1 has two states ($k = 2$). An

[2]These are related to the first comments in each m file, which are available by opening it in the editor. An alternative way to access the description of each file is to use the help function (e.g. `help('MS_Regress_Fit')`).

alternative representation is:

$$y_t = \mu_1 + \epsilon_t \qquad \text{for State 1} \tag{2}$$
$$y_t = \mu_2 + \epsilon_t \qquad \text{for State 2} \tag{3}$$

where:

$$\epsilon_t \sim (0, \sigma_1^2) \qquad \text{for State 1} \tag{4}$$
$$\epsilon_t \sim (0, \sigma_2^2) \qquad \text{for State 2} \tag{5}$$

This representation clearly implies two different processes for the dependent variable y_t. When the state of the world for time t is 1 (2), then the expectation of the dependent variable is μ_1 (μ_2) and the volatility of the innovations is σ_1^2 (σ_2^2).

For an empirical example, y_t can represent a vector of log returns for a financial asset[3]. The value of μ_1 is the expected return on a bull market state, which implies a positive trend for financial prices and consequently a positive log return for y_t. The lower value μ_2 measures the expected log return for the bear market state, which then implies a negative trend in prices.

The different volatilities (σ_1^2 and σ_2^2) in each state represent the higher uncertainty regarding the predictive power of the model in each state of the world. Going back to my example, one could expect that the bear market state is more volatile than the bull market. This implies that prices go down faster than they go up[4]. This means that we can expect σ_{Bear}^2 to be higher than σ_{Bull}^2.Note that I do not identify the states (e.g. bull market is state 1). In general, the S_t variable simply index the states, where the interpretation is given by looking at parameter's values.

So far I haven't said how exactly the switching from one state to the other happens. For instance, how is that one should know which state of the world is for each point in time. Suppose that we had assumed a deterministic transition of states where state 1 is true for time t

[3]The log return is the geometric change for the price of a particular asset (e.g. stock price) between time $t-1$ and t. Formally, if P_t is the price of a particular asset for time t, then $log(P_t/P_{t-1})$ is the log return for time t.

[4]The usual explanation for this effect is that traders react faster to bad news when comparing to good news. This can also be explained by the presence of limit loss orders, which will sell at market prices once a particular threshold in the prices has been breached. When used by a significant amount of traders and at different threshold levels, these limit loss orders will create a cascade effect, therefore accelerating the downfall of prices.

when a exogenous time series z_t is positive. This greatly simplifies the model as each state is observable and, therefore, we can treat the model given before as a regression with dummy variables. This would take the shape of $y_t = D_t(\mu_1 + \epsilon_{1,t}) + (1 - D_t)(\mu_2 + \epsilon_{1,t})$, where D_t is the dummy variable taking value of 1 if $z_t > 0$ and 0 otherwise.

For a markov regime switching model, the transition of states is stochastic (and not deterministic). This means that one is never sure whether there will be a switch of state or not. But, the dynamics behind the switching process is know and driven by a transition matrix. This matrix will control the probabilities of making a switch from one state to the other. It can be represented as:

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{k1} & \cdots & p_{kk} \end{bmatrix} \tag{6}$$

For 6, the element[5] in row i, column j (p_{ij}) controls the probability of a switch from state j to state i. For example, consider that for some time t the state of the world is 2. This means that the probability of a switch from state 2 to state 1 between time t and $t+1$ will be given by p_{12}. Likewise, a probability of staying in state 2 is determined by p_{22}. This is one of the central points of the structure of a markov regime switching model, that is, the switching of the states of the world is a stochastic process itself[6]. Usually these transition probabilities are assumed constant, but it is also possible to allow it to vary over time. This is called the TVTP (time varying transition probabilities) model, which is not supported[7] by `MS_Regress`. See Wang [2003] for more details in this type of specification.

4.1 Maximum likelihood estimation

A general MS model can be estimated with two different methods, maximum likelihood or Bayesian inference (Gibbs-Sampling). In the

[5]The ordering of the elements in the matrix is a matter of notation. It is not uncommon to find different notations in the literature.

[6]In fact, the probabilities of each regime over time can be represented as a AR process, [Hamilton, 1994]

[7]Zhuanxin Ding developed a matlab package for TVTP models based on MS_Regress. You can access it at: `http://www.mathworks.com/matlabcentral/fileexchange/37144`.

matlab package all of the models are estimated using maximum likelihood and this procedure is described here. Consider the following regime switching model:

$$\begin{aligned} y_t &= \mu_{S_t} + \epsilon_t & (7)\\ \epsilon_t &\sim N(0, \sigma^2_{S_t}) & (8)\\ S_t &= 1, 2 & (9) \end{aligned}$$

The log likelihood of this model is given by:

$$\ln L = \sum_{t=1}^{T} \ln \left(\frac{1}{\sqrt{2\pi\sigma^2_{S_t}}} \exp \left(-\frac{y_t - \mu_{S_t}}{2\sigma^2_{S_t}} \right) \right) \quad (10)$$

For the previous specification, if all of the states of the world were know, that is, the values of S_t are available, then estimating the model by maximum likelihood is straightforward. All you need is to maximize Equation 10 as a function of parameters μ_1, μ_2, σ^2_1 and σ^2_2. Is should be clear by now that this is not the case for a markov switching model, where the states of the world are unknown.

In order to estimate a regime switching model where the states are not know, it is necessary to change the notation for the likelihood function. Considering $f(y_t|S_t = j, \Theta)$ as the likelihood function for state j conditional on a set of parameters (Θ), then the full log likelihood function of the model is given by:

$$\ln L = \sum_{t=1}^{T} \ln \sum_{j=1}^{2} \left(f(y_t|S_t = j, \Theta) Pr(S_t = j) \right) \quad (11)$$

which is just a weighted average of the likelihood function in each state, where the weights are given by the state's probabilities. When these probabilities are not observed, we cannot apply equation 11 directly, but we can make inferences on the probabilities based on the available information. This is the main idea of Hamilton's filter, which is used to calculate the filtered probabilities of each state based on the arrival of new information.

Consider ψ_{t-1} as the matrix of available information at time $t-1$. Using Hamilton's filter the estimates of $Pr(S_t = j)$ are available using the following iterative algorithm:

1. Setting a guess for the starting probabilities ($t = 0$) of each state $Pr(S_0 = j)$ for $j = 1, 2$. Here one can use a naive guess, e.g.

$Pr(S_0 = j) = 0.5$ or the steady-state (unconditional) probabilities of S_t:

$$Pr(S_0 = 1|\psi_0) = \frac{1 - p_{11}}{2 - p_{11} - p_{22}} \tag{12}$$

$$Pr(S_0 = 2|\psi_0) = \frac{1 - p_{22}}{2 - p_{22} - p_{11}} \tag{13}$$

2. Set $t = 1$ and calculate the probabilities of each state given information up to time $t - 1$:

$$Pr(S_t = j|\psi_{t-1}) = \sum_{i=1}^{2} p_{ji}(Pr(S_{t-1} = i|\psi_{t-1})) \tag{14}$$

where p_{ji} are the transition probabilities from the markov chain (see equation 6).

3. Update the probability of each state with the new information from time t. This is accomplished by using the parameters of the model in each state, in this case μ_1,μ_2,σ_1,σ_2, the transition probabilities p_{11} and p_{22} for the calculation of the likelihood function in each state ($f(y_t|S_t = j, \psi_{t-1})$) for time t. After that, use the following formula to update the probability of each state given the new information:

$$Pr(S_t = j|\psi_t) = \frac{f(y_t|S_t = j, \psi_{t-1})Pr(S_t = j|\psi_{t-1})}{\sum_{j=1}^{2} f(y_t|S_t = j, \psi_{t-1})Pr(S_t = j|\psi_{t-1})} \tag{15}$$

4. Set $t = t + 1$ and repeat steps 2-3 until $t = T$, that is, you reached all observations in your sample. This should provide a set of filtered probabilities for each state, from $t = 1$ to $t = T$.

The previous set of steps provides the probabilities that one needs for calculating the log likelihood of the model as a function of the set of parameters:

$$\ln L = \sum_{t=1}^{T} \ln \sum_{j=1}^{2} (f(y_t|S_t = j, \Theta)Pr(S_t = j|\psi_t)) \tag{16}$$

The estimation of the model is obtained by finding the set of parameters that maximize the previous equation. While I used $k = 2$ in

the previous example, the formula for the generic case of k states is also available, where matrix notation can greatly simplify the calculations. See Hamilton [1994] and Kim and Nelson [1999] for further details on this topic.

One important point in the estimation of the regime switching model is that the parameters in the transition matrix are not variation free. Since they are probabilities, their values have to be between 0 and 1 and need to sum to 1 in each column of the transition matrix P. For $k = 2$, these conditions are easy to implement by just applying a numerical transformation[8] for p_{11} and p_{22} and setting $p_{12} = 1 - p_{11}$ and $p_{22} = 1 - p_{21}$. But, be aware that when $k > 3$ this procedure will not work as the second condition of the model will not be guaranteed.

One solution to this problem is to use a constrained optimization function, so that the desired properties in the parameter vector are explicitly defined in the search of the log likelihood solution. This was the approach used in the previous version of the code, with the use of Matlab's fmincon function. Recently Zhuanxin Ding provided an interesting numerical transformation[9] that made possible for the transition matrix to be estimated in a variation free framework. This allowed for the use of different optimization functions such as fminsearch and fminunc in the estimation of the model.

5 Interface of MS_Regress for estimation

When thinking in computational terms, a (univariate[10]) markov switching model can be represented in a generalized notation. Consider the following formula:

$$y_t = \sum_{i=1}^{N_{nS}} \beta_i x_{i,t}^{nS} + \sum_{j=1}^{N_S} \phi_{j,S_t} x_{j,t}^{S} + \epsilon_t \tag{17}$$

$$\epsilon_t \sim P(\Phi_{S_t}) \tag{18}$$

This representation can nest a high variety of univariate markov switching specifications. The terms N_S and N_{nS} simply counts the

[8] See Hamilton [1994].

[9] See Zhuanxin [2012].

[10] The package also has support for multivariate models but, for sake of simplicity, I restrict the introduction in the topic for univariate specifications, only.

number of switching (and non switching) coefficients, respectively. The variable $x_{i,t}^{nS}$ is a subset of $x_{i,t}$ and contains all explanatory variables which don't have a switching effect. Likewise, the variable $x_{j,t}^{S}$ contains all the variables that have the switching effect. The term $P(\Phi)$ is the assumed probability density function of the innovations, with its own set of parameters (vector Φ).

When translating the structure in 17 and 18 into a computational framework, the required information one would need in order to set up its own specification is the explained (y_t) and the explanatory data ($x_{i,t}$), the location (and number) of parameters that will switch states and the shape of the probability density function for the innovations. The MS_Regress package has a very intuitive way of addressing such structure, which makes it a handy package for this type of specifications.

The central point of this flexibility resides in the input argument `S`, which controls for where to include markov switching effects. The package can estimate univariate and multivariate markov switching models but, the interface is slightly different between the cases. Let's first begin with the univariate interface.

5.1 Interface to univariate modeling

When in MATLAB environment, the way to call the fitting function of the package is[11]:

```
1 Spec_Out=MS_Regress_Fit(dep,indep,k,S,advOpt)
```

The first variable `Spec_Out` is the output structure which contains all the information regarding the estimated model. Variable `dep` is the dependent variable and it can either be a vector (univariate) or a matrix (multivariate model). The input `indep` represents the independent variables. For the case of a univariate model, it is represented as a matrix with columns equal to the number of regressors. For the case of a multivariate model, this variable is a cell array[12]. The last three inputs, `k`, `S` and `advOpt` are, respectively, the number of states in the model, the locations of the switching parameters and advanced

[11] Special thanks to Florian Knorm for providing a latex package for Matlab code (available at `http://www.mathworks.com/matlabcentral/fileexchange/8015/`).

[12] Details on multivariate models are given later on the paper.

options fed to the algorithm. Further details on these are given next, with the help of an example.

Consider the following case of a model with two explanatory variables ($x_{1,t}$, $x_{2,t}$) where the innovations follow a Gaussian (Normal) distribution and the input argument `S`, which is passed to the fitting function MS_Regress_Fit.m, is equal to `S=[1 1 1]`. Given this configuration, the model is represented as:

$$y_t = \beta_{1,S_t} x_{1,t} + \beta_{2,S_t} x_{2,t} + \epsilon_t \tag{19}$$

$$\epsilon_t \sim N(0, \sigma^2_{S_t}) \tag{20}$$

Where:

S_t: The state at time t, that is,$S_t = 1..k$, where k is the number of states.
$\sigma^2_{S_t}$: The variance of the innovation at state S_t.
β_{i,S_t}: Beta coefficient for explanatory variable i at state S_t where i goes from 1 to 2.
ϵ_t: Residual vector which follows a particular distribution (in this case Normal).

Now, changing the input argument to `S=[1 1 0]`, the model is:

$$y_t = \beta_{1,S_t} x_{1,t} + \beta_{2,S_t} x_{2,t} + \epsilon_t \tag{21}$$

$$\epsilon_t \sim N(0, \sigma^2) \tag{22}$$

Notes that that with `S=[1 1 0]`, the variance term is no longer switching states. Now, with the switching argument as `S=[0 1 1]`, the model is:

$$y_t = \beta_1 x_{1,t} + \beta_{2,S_t} x_{2,t} + \epsilon_t \tag{23}$$

$$\epsilon_t \sim N(0, \sigma^2_{S_t}) \tag{24}$$

With this change in the input argument `S`, only the second coefficient and the model's variance are switching according to the transition probabilities. That is, the first coefficient (β_1) does not change states. Therefore, the logic is clear: the first elements of S control the switching dynamic of the mean equation, while the last term controls the switching dynamic of the residual vector, including distribution

parameters. For an example with extra distribution parameters, consider the following definition for a model with GED (generalized error distribution) innovations and with input `S=[1 1 1 1]`. This configuration yields:

$$y_t = \beta_{1,S_t} x_{1,t} + \beta_{2,S_t} x_{2,t} + \epsilon_t \tag{25}$$

$$\epsilon_t \sim GED(0, \sigma^2_{S_t}, K_{S_t}) \tag{26}$$

In this setup, the new parameter K will also switch states. This coefficient is part of the GED distribution and should not be confused with the k (number of states in the model). If we had set `S=[1 1 1 0]`, the model would switch in all coefficients, except in the K parameter.

As an example for the markov switching fitting function, assume that there is a variable called `logRet` in `MATLAB`'s workspace. This represents the log returns of a particular asset over a particular frequency (e.g. daily). Consider the input of the following options at `MS_Regress_Fit()`:

```
% Defining inputs
dep=logRet;
constVec=ones(length(dep),1);
indep=constVec;
k=2;
S=[1 1];
advOpt.distrib='Normal';
advOpt.std_method=1;

% Calling fitting function
Spec_Out=MS_Regress_Fit(dep,indep,k,S,advOpt)
```

For the last piece of code, the vector `dep` is the dependent variable. The term `constVec` is a vector full of ones (the constant), k is the number of states and S defines the location of the switching parameters. The model represented in computational terms in the last piece of `Matlab` code is equivalent to the model with only a constant term and two states given previously in the paper (see Equation 1).

The input structure `advOpt` determines advanced information of the model, in this case, the distribution assumption and the method for calculating the standard errors. More details regarding the use of `advOpt` can be found in a later section of the paper.

The inputs given before are related to the estimation of the following equations:

$$\text{State 1 } (S_t = 1) \tag{27}$$

$$y_t = \beta_1 + \epsilon_t \tag{28}$$

$$\epsilon_t \sim N(0, \sigma_1^2) \tag{29}$$

$$\text{State 2 } (S_t = 2) \tag{30}$$

$$y_t = \beta_2 + \epsilon_t \tag{31}$$

$$\epsilon_t \sim N(0, \sigma_2^2) \tag{32}$$

with:

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{2,1} \\ p_{1,2} & p_{2,2} \end{pmatrix} \tag{33}$$

as the transition matrix, which controls the probability of a switch from state j (column j) to state i (row i). The sum of each column in P is equal to one, since they represent full probabilities of the process for each state.

Another flexibility of the package is that, since I wrote it for dealing with a generic type of regression, you can set any kind of explanatory variable in the model as long as they are observed (you have it available for the whole time period studied). This includes autoregressive components, constants or just plain regression on other variables.

If you run the script `Example_MS_Regress_Fit.m` with `MATLAB` version 7.10.0.499 (R2010a), this is the output you should be getting if you have all the proper packages installed (optimization, statistics).

```
***** Numerical Optimization Converged *****

Final log Likelihood: 2487.232
Number of estimated parameters: 8
Optimizer: fminsearch
Type of Switching Model: Univariate
Distribution Assumption -> Normal
Method SE calculation -> 1

***** Final Parameters for Equation #1 *****

---> Non Switching Parameters <---

Non Switching Parameter for Eq #1, Indep column 2
     Value:                 0.4788
     Std Error (p. value): 0.0269 (0.00)
Non Switching Parameter for Eq #1, Indep column 3
     Value:                 0.1555
     Std Error (p. value): 0.0333 (0.00)

---> Switching Parameters (Dist Parameters)  <---

State 1
    Model's Variance:      0.000266
    Std Error (p. value):  0.0000 (0.00)
State 2
    Model's Variance:      0.000837
    Std Error (p. value):  0.0001 (0.00)

--->   Switching Parameters (Regressors)  <---

Switching Parameters for Eq #1 - Indep column 1

State 1
   Value:                0.0003
   Std Error (p. value): 0.0006 (0.63)
State 2
   Value:                0.0005
   Std Error (p. value): 0.0016 (0.75)

---> Transition Matrix (std. error, p-value) <---

      0.99 (0.20,0.00)   0.02 (0.21,0.91)
      0.01 ( NaN, NaN)   0.98 ( NaN, NaN)

```

```
 ---> Expected Duration of Regimes <---

      Expected duration of Regime #1: 109.01
      Expected duration of Regime #2: 43.73
```

The figures generated by the code are as follows:

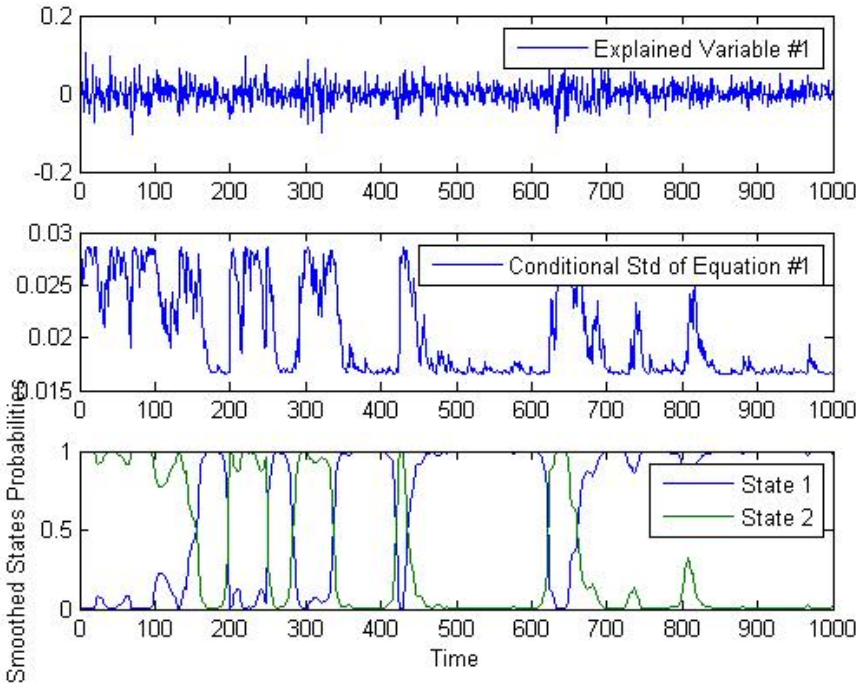

Figure 1: Output from `Example_MS_Regress_Fit.m`

5.2 Interface to multivariate modeling

By now it should be clear the use of input argument `S`. The multivariate interface I built for this package has the same intuition as in the univariate case and consists of basically a group of inputs in cell array notation, where the elements iterates over the equations in the system. An example should make it clear. Consider the following model:

$$y_{1,t} = \beta_{1,1,S_t} * x_{1,t} + \beta_{1,2,S_t} * x_{2,t} + \epsilon_{1,t} \tag{34}$$

$$y_{2,t} = \beta_{2,1,S_t} * x_{3,t} + \epsilon_{2,t} \tag{35}$$

with

$$\epsilon_{1,t} \sim N(0, \sigma^2_{1,S_t}) \tag{36}$$

$$\epsilon_{2,t} \sim N(0, \sigma^2_{2,S_t}) \tag{37}$$

$$S_t = 1, 2 \tag{38}$$

$$cov(\epsilon_{1,t}, \epsilon_{2,t}) = 0 \tag{39}$$

This last system of 2 equations is translated in the package's notation as:

```
% Defining input variables
dep=y;
indep{1}=[x1 x2];
indep{2}=x3;
k=2;
S{1}=[1 1 1];
S{2}=[1 1];

% Calling fitting function
Spec_Out=MS_Regress_Fit(dep,indep,k,S);
```

For the last piece of code, variable `y` is a matrix with two columns. As one can see, the inputs have the same shape as for the univariate case, but they are iterated over the equations of the system by using cell arrays[13]. Notice that in the previous code it is not allowed covariance between the residuals of the different equations (see 39). This is a default option of the algorithm, but it can be changed so that a full covariance matrix is estimated from the data.

All options to univariate modeling, including reduced estimation, are also available for the multivariate case. For further details see the script `Example_MS_Regress_Fit_MultiVar.m`, which can be found in the package's zip file.

[13]For those not familiarized with Matlab, cell arrays are a type of flexible structure that can accommodate any kind of data. They are similar to the use of 'lists' in R.

5.3 Estimating autoregressive Models (MS - VAR)

The package also comes with a simple wrapper function for estimating a general autoregressive markov switching model. In the current version of the package, moving average terms and error correction models are not supported.

For example, consider a matrix consisting of two time series ($Y_t = [y_{1,t}\ y_{2,t}]$) that follows an autoregressive system with B_{S_t} as the matrix of parameters:

$$Y_t = B_{S_t} Y_{t-1} + \epsilon_t \tag{40}$$

with:

$$\epsilon_t \sim N(0, \Sigma_{S_t}) \tag{41}$$

$$\Sigma_{S_t} = \begin{pmatrix} \sigma^2_{1,S_t} & \sigma_{1,2,S_t} \\ \sigma_{1,2,S_t} & \sigma^2_{2,S_t} \end{pmatrix} \tag{42}$$

This model translates into the package's notation as:

```
% Defining input
dep=logRet(:,1:2);
nLag=1;
k=2;
advOpt.diagCovMat=0;
doIntercept=0;

% Calling fitting function
Spec_Out=MS_VAR_Fit(dep,nLag,k,doIntercept,advOpt);
```

As one can see, the input structure is similar to `MS_Regress_Fit`. The new inputs, `nLag`, `doIntercept` and `advOpt.diagCovMat` are respectively the number of lags in the system, the choice of using intercept in the equations and the use of a full matrix as the covariance matrix[14].

[14]This implies that all elements in the covariance matrix are estimated from the data. If `advOpt.diagCovMat=1`, then only the elements in the diagonal (the variances) are estimated and the rest (the non diagonal elements, the covariances) are all set to zero.

5.4 The output from the estimation function

The estimation function `MS_Regress_Fit` returns a structure with all the information regarding your model. This comprises of the following fields:

- Coeff: A Structure with the fields:
  - Coeff.p: The transition probability matrix
  - Coeff.nS_Param: All non switching betas (iterated over equations (cells) and independent variables (rows))
  - Coeff.S_Param: All switching betas (iterated over equations (cells), over independent variables (rows) and states (columns))
  - Coeff.covMat: Covariance matrix (iterated over states (cell)).
- filtProb: The filtered probabilities of regimes (iterated over the states (columns))
- LL: Final log likelihood of model
- k: Number of states
- param: All estimated parameters in vector notation
- S: Switching flag control (iterating over equations (cell))
- advOpt: advanced options fed to algorithm (see next section for details)
- condMean: Conditional mean calculated[15] by the model
- condStd: Conditional standard deviation calculated[16] by the model.
- resid: Residuals from the model (iterating over equations (columns))
- stateDur: Expected duration of each state
- smoothProb: Smoothed probabilities of regimes (iterated over the states (columns))
- nObs: Number of observations (rows) in the model
- Number_Parameters: Number of estimated parameters
- Coeff_SE: A structure with all standard errors of coefficients (same fields as Coeff)

[15] These are calculated based only on filtered probabilities prior to time t.
[16] These are also calculated with information prior to time t.

- Coeff_pValues: A structure with all parameter's p-values (same fields as Coeff)
- AIC: Akaike information criteria of the estimated model
- BIC: Bayesian information criteria for estimated model

These fields should contain all the information you need for further testing of the model. If you're missing something, please let me know.

6 Interface of MS_Regress for simulation

While the most likely use of MS_Regress is for the estimation of a regime switching model based on a time series data, I also provide functions for simulation of a model, which can be used in a teaching environment.

The structure of the interface of the simulation function `MS_Regress_Sim` is very similar to the fitting function `MS_Regress_Sim`, but there are some differences. Here I make some remarks in the use of `MS_Regress_Sim`:

- In the use of `MS_Regress_Sim`, the vector S is related only to the switching dynamic of the conditional mean. This is different than the notation in `MS_Regress_Fit`, where the last elements of S defined the switching or not of the innovation's parameters. In the simulation function it is assumed that the conditional variance is always switching states.
- The independent variable used in the simulation are created within the code as normal random variables with user defined moments (see input `Coeff.indepMean` and `Coeff.indepStd`).
- The input variable `Coeff.nS_param` must always be defined, even when it is not switching states.

An example should make it clear. Consider the following model:

$$y_t = 0.5 + \epsilon_t \quad \text{State 1} \tag{43}$$
$$y_t = -0.5 + \epsilon_t \quad \text{State 2} \tag{44}$$
$$\epsilon_t \sim N(0, 0.5^2) \quad \text{State 1} \tag{45}$$
$$\epsilon_t \sim N(0, 1) \quad \text{State 2} \tag{46}$$

$$P = \begin{bmatrix} 0.95 & 0.1 \\ 0.05 & 0.9 \end{bmatrix} \tag{47}$$

In order to simulate the previous set of equations, one need to build the innovations in each state and also the states of the world over time, which are defined according to the transition matrix. The simulation of the previous model is accomplished with the following code[17] in `MS_Regress_Sim`:

```
% Example Script for MS_Regress_Simul.m

% add 'm_Files' folder to the search path
addpath('m_Files');

clear; clc;

nr=500;  % Number of observations in simulation
advOpt.distrib='Normal';   % Distribution assumption
% Transition matrix (this also defines the value
% of k)

Coeff.p=[.95 .1; ...
         .05 .9];

% Setting up which variables at indep will have
%switching effect
Coeff.S=[0 1];

% Setting up the coefficients at non switching
%parameters (each row is each % variables
%coefficient). The order is the same as Coeff.S

% Setting up the coefficients at non switching
%parameters
Coeff.nS_param=0;

% Setting up the coefficients at non switching
% parameters (each row is each variables coefficient
% and each collum is each state). This example has
% 1 switching parameter and 2 states

Coeff.S_param(1,1)= .5;
```

[17] This script is available in the zip file as `Example_MS_Regress_Sim`.

```
34 Coeff.S_param(1,2)=-.5;
35
36 % Setting up the standard deviavion of the
37 %model at each state
38 Coeff.Std(1,1)=0.5;
39 Coeff.Std(1,2)=0.1;
40
41 % The explanatory variables used in the simulation
42 % are always random normal, with
43 % specific mean and standard deviation
44
45 Coeff.indepMean=[1 0];
46 Coeff.indepStd= [0 0];
47
48 % getting the value of k, according to Coeff.p
49 k=size(Coeff.p,1);
50
51 % calling simulation function
52 [Simul_Out]=MS_Regress_Sim(nr,Coeff,k,advOpt.distrib);
53
54 rmpath('m_Files');
```

The inputs to the simulation function `MS_Regress_Sim` are explained within the code's comments. When running the previous script, a figure with the time series plot of the simulated model will be created:

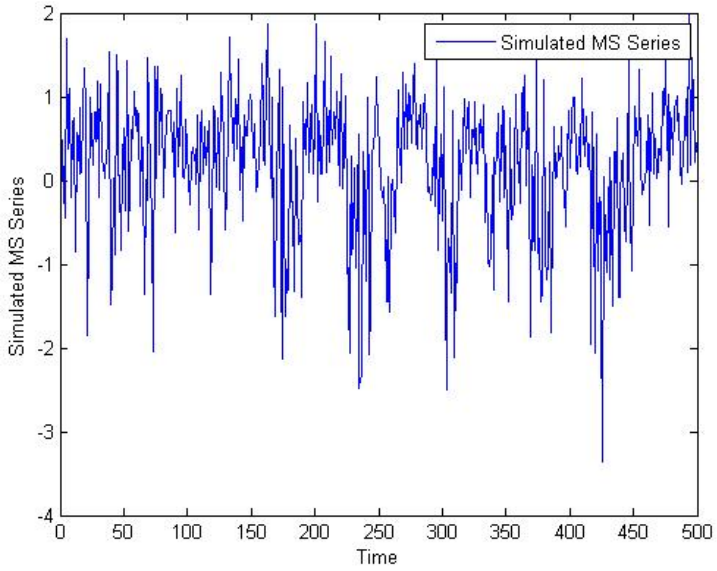

Figure 2: Output from `Example_MS_Regress_Sim.m`

This plot will be different for each run of the code. For further instruction on how to use the simulation function in the case of more complex models, including the multivariate case, have a look in the examples scripts provided within the package's zip file.

7 Using advanced options

When I was writing this markov switching package I always had in mind that it should be simple and intuitive to use but, at the same time, be complete in providing different options to the user. In the use of the fitting function `MS_Regress_Fit.m`, all of the different options of the algorithm are controlled by the input `advOpt`. The possible fields are:

advOpt.distrib: Defines the distribution to be used in the maximum likelihood calculation. If `advOpt.distrib='Normal'`, then the distribution used is the Gaussian, if `advOpt.distrib='t'`, it is used the t distribution, with one extra parameter (degrees of freedom). If this field is equal to `'GED'` then the distribution for the error is the

Generalized Error distribution with one extra parameter (K).

advOpt.std_method - Defines the method to be used for the calculation of the standard errors of the estimated coefficients. The options are:

- advOpt.std_method=1: Calculation of the covariance matrix is performed using the second partial derivatives of log likelihood function (a.k.a. the Hessian matrix).
- advOpt.std_method=2: Calculation of the covariance matrix is performed using the first partial derivatives of log likelihood, that is, the outer product matrix. This methods approximates the Hessian matrix with the gradient vector and is a robust choice for when `std_method=1` fails to give reasonable values[18]. See Hamilton [1994] for more details in the calculations of standard errors.

advOpt.useMex: Defines whether to use the mex[19] version of Hamilton's filter in the calculation of likelihood function. The advantage in using the mex version is the increase of speed in the fitting function. But, in order to use it, you'll need a proper C++ compiler. Please check next topic for more details.

advOpt.constCoeff: Defines the use of constrained/reduced estimation of the model. This is particularly useful when the desired model has a very particular representation which cannot be addressed in the usual notation. See the following topic for instructions on how to use this feature.

advOpt.diagCovMat: Defines the use of a diagonal matrix for sigma (covariance matrix) in a multivariate estimation, only. If `advOpt.diagCovMat=0` then the whole covariance matrix is estimated from the data.

[18]The calculation of the Hessian by numerical differentiation is not guaranteed to provide a real number. When this happens, the function `MS_Regress_Fit` will output `NaN` (Not a Number) values for the standard errors.

[19]Mex files stands for matlab's executable file and is a way to interface `MATLAB` with low level programming languages such as `C++` and `Fortran`.

advOpt.optimizer: Defines which Matlab's optimizer to use in the maximum likelihood estimation of the model.[20]

advOpt.printOut: Flag for printing out to screen the model in the end of estimation.

advOpt.printIter: Flag for printing out numerical iterations of maximum likelihood estimation.

advOpt.doPlots: Flag for plotting fitted conditional standard deviations and smoothed probabilities in the end of estimation.

Next in Table 1 I show the possible values for each input in advOpt and also the default values (if no value is assigned).

Input Argument	**Possible Values**	**Default Value**
advOpt.distrib	'Normal','t' or 'GED'	'Normal'
advOpt.std_method	1 or 2	1
advOpt.useMex	1 or 0	0
advOpt.constCoeff	Any number or string 'e' (see next topic for details)	A structure with fields containing string 'e' (all coefficients are estimated from the data)
advOpt.diagCovMat	1 or 0	1
advOpt.optimizer	'fminsearch','fmincon' or 'fminunc'	'fminsearch'
advOpt.printOut	1 or 0	1
advOpt.printIter	1 or 0	1
advOpt.doPlots	1 or 0	1

Table 1: Default Values for input arguments in advOpt.

[20]Special thanks for Zhuanxin Ding for providing a clever numerical transformation in the transition matrix that allows for the model to be estimated without the need of equality constraints in the optimization, that is, using functions 'fminsearch' and 'fmincon'. See Zhuanxin [2012] for details.

7.1 Using the mex version of Hamilton's filter

The filter behind the calculation of the filtered probabilities of the model is computationally intensive and this may be a pain when dealing with large amounts of data. The MS_Regress package has a mex version of the likelihood function that performs the heavy duty of the model, Hamilton's filter, in a cpp mex file. Depending on the number of observations in the system, the gain in speed can be higher than 50%, meaning that the mex version of the filter can decrease in half the time needed to estimate the model.

In order to use it, first you'll have to compile the C version of the filter. The file in question is mex_MS_Filter.cpp, which is located at the folder m_Files. The cpp file was compiled and tested using MATLAB 2010a and MS VC 2008. You can get the last one freely on the internet[21]. After installing it, type in MATLAB the command:

```
mex -setup;
```

This will take you to a set of steps for configuring your default mex compiler to MS VC 2008. After that, just use the command:

```
mex mex_MS_Filter.cpp;
```

in the directory m_Files and try running the script Example_MS_Regress_Fit_with_MEX.m.

Please note that the .cpp function will NOT compile under MATLAB's native LCC. For others C++ compilers, I haven't tested it, hopefully it will work. If you successively compiled the code with other compiler, please let me know.

If you're having problems compiling it, please email me and I'll send you the compiled mex file which should work for Matlab 32bit version.

7.2 Using advOpt.constCoeff (constraining coefficients)

The MS_Regress package also supports constrained/reduced estimation of the model. This means that, for all the coefficients, you can

[21] http://www.microsoft.com/express/

choose whether you want to estimate the parameter from the data or if you want to fix it to a specific value. This feature also holds for multivariate models.

Consider the following case: you know for sure (or at least have a theory) that one (or more) of the switching coefficients (say column 2 of indep) has the value of 0.5 when the model is in state 1 and value of -0.5 when the model is in state 2. You now want to know what are the maximum likelihood estimates for the other coefficients given this restriction. The package allow for this particular specification (or any other as matter of fact) to be set. The interface for using constrained estimation has the following principles:

1. The parameters are grouped with the notation:
   - nS_Param: All non switching coefficients at indep matrix (which were chosen with argument S)
   - S_Param: All switching coefficients at indep matrix (also chosen with input S)
   - p: Transition matrix
   - $covMat$: Covariance matrix of innovations
   - df: Degrees of freedom for t distribution (if applicable)
   - K: Parameter for GED distribution (if applicable)
2. The size of each of these fields are set according to the number of switching/non switching parameters and the number of states in the model (value of k). For all of them (except $covMat$), the cells iterates over the equations in the system, the columns iterate the states and the rows iterate the coefficients. For instance, for `nS_Param{iEq}`, the element (1,1) of this matrix is the first non switching parameters of equation iEq, the element (2,1) is the second non switching parameter and so on. For `S_Param{iEq}`, the element (1,2) is the parameter of the first switching variable, at state 2, equation iEq. The sizes of `covMat`, `df` and `K` also have to respect the choices made at input `S`. For instance if the model is switching in the standard deviation of the innovations, then `covMat` should have size {1,k}, otherwise it should be a {1,1} cell. For the case of multivariate models, `covMat` should be sized according to the number of equations in the system.
3. The rule for building the fields is, use the string `'e'` (as in 'estimate') if you want the respective parameter to be estimated from

data or a numeric value if you want the respective parameter to take that value. For instance, using:

```
advOpt.constCoeff.S_Param{1}={0.5 , 'e' ; ...
                              'e' , 0.1};
```

you're telling `MS_Regress_Fit` to set the coefficients of the first switching parameter of equation 1, in state 1, equal to 0.5, the second switching parameter, state 2, equal to 0.1 and estimate the rest of them.

Examples should make the structure on the use of `advOpt.constCoeff` clear. Consider the estimation of the following model:

$$\text{State 1 } (S_t = 1) \tag{48}$$

$$y_t = \beta_1 + 0.x_{1,t} + \beta_{2,1}x_{2,t} + \epsilon_t \tag{49}$$

$$\epsilon_t \sim N(0, \sigma_1^2) \tag{50}$$

$$\text{State 2 } (S_t = 2) \tag{51}$$

$$y_t = \beta_1 + \beta_{1,2}x_{1,t} + 0x_{2,t} + \epsilon_t \tag{52}$$

$$\epsilon_t \sim N(0, \sigma_2^2) \tag{53}$$

with:

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{2,1} \\ p_{1,2} & p_{2,2} \end{pmatrix} \tag{54}$$

For the previous specification, the restrictions are:

$\beta_{1,1} = 0$
$\beta_{2,2} = 0$

In order to estimate this model, these are the options fed to the fitting function `Example_MS_Regress_Fit_using_constCoeff.m`:

```
% Defining Inputs
k=2;
S=[0 1 1 1];
advOpt.distrib=`Normal';
advOpt.std_method=1;
```

```
6
7  % Defining constrained parameters
8  advOpt.constCoeff.nS_Param{1}={'e'};
9  advOpt.constCoeff.S_Param{1}={ 0 ,'e' ; 'e', 0}
10
11 advOpt.constCoeff.covMat{1}(1,1)={'e'};
12 advOpt.constCoeff.covMat{2}(1,1)={'e'};
13
14  advOpt.constCoeff.p={'e','e' ; 'e','e' };
15
16 % Estimate resulting model
17 Spec_Out=MS_Regress_Fit(dep,indep,k,S,advOpt);
```

For the previous code, the command

```
advOpt.constCoeff.nS_Param{1}={'e'};
```

is telling the function `MS_Regress_Fit` to estimate the non switching parameter of the first (and only) equation in the system. The input

```
advOpt.constCoeff.S_Param{1}={ 0 ,'e'; 'e', 0};
```

fix the switching parameter of indep column 2 to 0 at state 1, indep column 3 equal to 0 at state 2 and estimate the rest of it. The commands:

```
advOpt.constCoeff.covMat{1}(1,1)={'e'};
```

and

```
advOpt.constCoeff.covMat{2}(1,1)={'e'};
```

defines the estimation of the covariance matrix for both states. The last input:

```
advOpt.constCoeff.p={'e','e'; 'e','e'};
```

directs the fitting function to estimate all of the transition probabilities.

Consider now the next example, where the model is:

$$\text{State 1 } (S_t = 1) \tag{55}$$

$$y_t = \beta_1 + 0.5x_{1,t} + \beta_{2,1}x_{2,t} + \epsilon_t \tag{56}$$

$$\epsilon_t \sim N(0, 0.0004) \tag{57}$$

$$\text{State 2 } (S_t = 2) \tag{58}$$

$$y_t = \beta_1 + \beta_{1,2}x_{1,t} - 0.8x_{2,t} + \epsilon_t \tag{59}$$

$$\epsilon_t \sim N(0, \sigma_2^2) \tag{60}$$

with:

$$\mathbf{P} = \begin{pmatrix} 0.95 & p_{2,1} \\ 0.05 & p_{2,2} \end{pmatrix} \tag{61}$$

Note that this model has the following restrictions:

$\beta_{1,1} = 0.5$
$\beta_{2,2} = -0.8$
$\sigma_1^2 = 0.0004$
$p_{1,1} = 0.95$
$p_{1,2} = 0.05$

The input arguments for `advOpt.constCoeff` that represent this model in `MS_Regress_Fit` are:

```
% Defining inputs
k=2;
S=[0 1 1 1];
advOpt.distrib='Normal';
advOpt.std_method=1;

% Defining restrictions
advOpt.constCoeff.nS_Param{1}={'e'};
advOpt.constCoeff.S_Param{1}={0.5, 'e' ; 'e',-0.8};
advOpt.constCoeff.covMat{1}(1,1)={0.0004};
advOpt.constCoeff.covMat{2}(1,1)={'e'};
advOpt.constCoeff.p={0.95,'e' ; 0.05,'e'};
```

As one can see, any parameter of the model can be constrained to a particular value, including the transition matrix[22].

[22]Be aware that when constraining the transition matrix, the probabilities have to sum to one in each column.

8 Comparing results against Hamilton [1989]

The markov switching specification of Hamilton [1989] is naturally a benchmark in this class of models and I get an unusual amount of emails regarding matching the results of the paper by using my package. This section will shed some lights in this issue.

The markov switching model of Hamilton [1989] is specified as:

$$y_t - \mu_{S_t} = \sum_{i=1}^{4} \phi_i (y_{t-i} - \mu_{S_{t-i}}) + \epsilon_t \tag{62}$$

$$\epsilon_t \sim N(0, \sigma^2) \tag{63}$$

As you can see from previous equation, the independent variables are conditional on the states, an unobserved process. That means that the regressors, $\sum_{i=1}^{4} \phi_i (y_{t-i} - \mu_{t-i})$, are also unobserved prior to estimation. My package was not built to deal with this type of setup but a two step estimation process is identifiable. Consider the following notation for Hamilton's Model:

$$z_t = y_t - \mu_{S_t} \tag{64}$$

This is equivalent to:

$$z_t = \sum_{i=1}^{4} \phi_i z_{t-i} + \epsilon_t \tag{65}$$

By rearranging 62 we get:

$$y_t = \mu_{S_t} + z_t \tag{66}$$

where z_t are the residuals from this reduced model. The two steps for the approximation which can be used to estimate Hamilton's model are:

1. Estimate using `MS_Regress_Fit`:

$$y_t = \mu_{S_t} + \epsilon_t \tag{67}$$

$$\epsilon_t \sim N(0, \sigma^2) \tag{68}$$

$$S_t = 1, 2 \tag{69}$$

2. Retrieve the $\hat{\epsilon_t}$ vector and regress it on four lags:

$$\hat{\epsilon_t} = \sum_{i=1}^{4} \beta_i \hat{\epsilon}_{t-i} + \nu_t \tag{70}$$

$$\nu_t \sim N(0, \sigma^2_{\nu_t}) \tag{71}$$

Note that the parameter $\sigma^2_{\nu_t}$ from last equation will approximate σ^2 of Hamilton's model. Next, I show the estimated parameters[23] from Hamilton [1989] and the parameters from my approximation:

[23]The GNP data is available at `http://weber.ucsd.edu/~jhamilto/software.htm`

Parameter	Hamilton [1989]	Two-step MS_Regress_Fit
μ_1	1.16	1.101
μ_2	-0.36	-0.48
$p_{1,1}$	0.9	0.906
$p_{2,2}$	0.75	0.682
σ^2	0.866	0.988
ϕ_1	0.01	0.126
ϕ_2	-0.06	0.104
ϕ_3	-0.25	-0.133
ϕ_4	-0.21	-0.104

Table 2: Comparison of parameter's estimates from Hamilton [1989]. The same model and data is fitted by a two step procedure with function `MS_Regress_Fit`. The estimates of the coefficients are then compared to the ones in the paper.

From Table 2, one can see that the parameters are fairly comparable for the markov regime switching part of the model. For the autoregressive part, they are not as comparable as for the first part but, since most of the ϕ_i are not statistically significant, the approximation performance is still good. When looking at the regime's probabilities (Hamilton [1994], page 697), it is also clear that they are very similar. The code for this estimation is available within the package's zip file (`Script_Hamilton_Comparison.m`).

9 Advises for using the package

You probably want to apply the package to your own time series. This topic will set some advices for using the fitting function.

1. For a better convergence, always check the scale of your dependent and independent variables. For instance, lets say the explained variable varies from -0.1 to 0.1, while one of the independent varies from -1000 to 1000. If you estimate with this setup (without any correction) the algorithm may not converge well[24]. In this case just divide the corresponding independent

[24]This is usually associated with the algorithm converging to -Inf in the log likelihood

variable by 1000 so that they are correctly scaled. This gentleman's (linear) transformation will help the optimization functions (fminsearch, fminunc or fmincon) to find the set of maximum likelihood parameters.

2. Always try to estimate simple models. For instance, don't try to estimate any model with $k > 3$ and number of explanatory variables higher than 4. The model's size (number of parameters) grows exponentially as n and k grows. For instance, in a univariate framework, if $k = 5$, $n = 4$ (4 explanatory variables) and you're switching in all coefficients, then the model has 50 parameters to be estimated from data, which is definitely too much for a gradient descent method (fmincon function). Don't get me wrong, the package will try to estimate it, but the solution is probably a local maximum and you can't really trust the output you get.
3. If using constrained estimation, make reasonable choices for the restricted coefficients, otherwise the function will have a hard time in finding the maximum likelihood estimates.
4. With the last update in the package, the choice for the optimizer to be used in the maximum likelihood was made available. If you are having problems in the estimation, try to change the optimizer and check whether the problem still persists.

If after those steps you're still having problems converging to a solution, send me an email with a nice personal introduction and an attached zip file containing:

- The scripts you're running (the main .m file). This should be send with all auxiliary m-files and data. The main program should run without any problem except the one you're reporting.
- The error message (if there is one, could be in .txt or just in the email scope).

10 Possible error messages

The code in `MS_Regress_Fit` is not guaranteed to run flawlessly for all versions of Matlab and the quality of the input data can also affect the estimation. With time (and lots of emails!) I was able to map out

function.

the error messages that people can get when running the code. Next, Table 3 you can find these messages, along with respective causes and solutions. If you have an error message which is not there, please let me know and I'll add it to the table.

Error Message	**Cause**	**Solution**
FMINCON/FMINSEARCH cannot continue because user supplied objective function failed with the following error: Variable 'indep_nS' is used as a command function.	This is an issue with your matlab version, which is not accepting the notation used in the newer versions.	Get a newer version of Matlab.
Error using svd. Input to SVD must not contain NaN or Inf.	This is an issue that usually appears when working with optimizer fmincon. Basically the optimizing function got stuck in a particular value of the log likelihood function and was not able to get out on its own. Usually this means that the input data is badly shaped (e.g. outliers) or the case of bad starting coefficients.	One simple solution which I have seen to work quite a lot is to divide the dependent variable by 100. This simple linear transformation will make the log likelihood function smoother and will help the optimizer in finding the maximum likelihood solution. Another solution which I also have seen to work is to look for outliers and removing (or smoothing) then from the data. At last, if none of them worked, try changing the optimizer to fminsearch or fminunc.

Table 3: Possible error messages, causes and solutions when using function MS_Regress_Fit.

11 Frequently asked questions

The first version of MS_Regress was published in August of 2007. Since then I have been receiving a significant amount of questions by email. Some of these questions have a tendency to reappear over time. Here I document the most common cases.

Q: Can I use your code for teaching and researching?

A: Yes! Fell free to use it or change it in any way you find best. My only request is that you provide the original source, citing the paper available at SSRN (see next topic, "Citing the Package").

Q: I am trying to replicate the results of a particular paper and the parameters I'm finding with MS_Regress do not match the ones reported in the article. What's the problem?

A: First, make sure that the data and the specification are matching. For example, if the paper uses a dataset from 2010-2011 and you use it from 2008-2011, the estimates will naturally be different. This is also true if the underlying model is different. But, if you do match data and the model then you should be aware that the estimation of the MS model is related to a optimizing algorithm finding a maximum likelihood estimate. For the same data and specification, different algorithms (or different software) will find different solutions. The divergence should be small, but it will still be there.

Q: I need to apply a particular statistical test in the post estimation of my model. Is there a function for such?

A: The package does not provide functions for hypothesis testing. But in the output of the estimation function I made sure that the user gets a lot of information so that he can apply any test on its own.

Q: The example scripts are not running in my computer. Can you check whats wrong?

A: It depends. Before you email me, please make sure that this is not something in the use of Matlab itself. The help files of Matlab are a good source of material for learning to run scripts and functions. If

you still can't figure it out, perhaps it is a version issue. Try the code in a newer version of Matlab and check how it goes. If still there are problems, fell free to email me.

Q: I'm rewriting some parts of MS_Regress but I'm having problems with the programming part. Can you help me with the codes?

A: Sorry but no. Helping people with programming in Matlab turned out to have a high cost in my side. If I helped everyone I would not have time to do my normal job. While I won't help you directly with the coding, I am willing to give you with some general guidelines of how (and where) to change the code.

Q: I'm estimating my model with MS_Regress and and the optimization converges. However, I noticed that some of the standard error and p-values are NaN (not a number). Why is that?

A: If you are finding NaN values for standard error or p-value, then it is because the numerical algorithm that calculates the first and/or second derivative has failed at that particular part. My advice is to either use an alternative optimization function (e.g. *fmincon*) or change the method that calculates the standard errors.

But, there a specific case where NaN are expected. If you are estimating the model with *fminsearch* or *fminunc* then the last row of the transition matrix is not estimated, but calculated indirectly from the other probabilities. Therefore it is not possible to calculate the first/second derivative and the standard error and p-value are not available.

Q: Are there other versions of the package written in different programming languages?

A: Yes, there are. Some time ago I wrote a version of MS_Regress for R. The package can be downloaded here:

`https://r-forge.r-project.org/projects/rmetrics/`

or installed with the following command in R:

install.packages("fMarkovSwitching", repos="http://R-Forge.R-project.org")

But be aware that given time constraints, I'm no longer maintaining this package.

Q: Are you interested in working in a consultancy job?

A: Yes. Fell free to drop me an email and we will discuss it.

12 Reporting a bug

I'm very happy to hear about any sort of bug in the program. If you have found one please report it to my email containing:

- Name of functions and lines of alleged bug (if applicable).
- Reason why you think it is a bug.
- Zip file with codes you're running (including data and scripts).
- `MATLAB` 's error message (if applicable).

And I'll happily look into it.

13 Citing the package

If you have used the package for research or teaching, make sure you cite the code, not just for acknowledgment but also for replication of results. My suggested citation is to use the paper from SSRN[25]:

Perlin, M. (2014) MS_Regress - The MATLAB Package for Markov Regime Switching Models. Available at SSRN: `http://ssrn.com/abstract=1714016` or `http://dx.doi.org/10.2139/ssrn.1714016`

14 Final remarks

The interest of this paper was in presenting the main features of the `MATLAB` package `MS_Regress`. As one can see, the interface of the

[25] Available in `http://ssrn.com/abstract=1714016`.

software is quite intuitive and it should be flexible enough to handle personalized markov switching specifications without any change in the original code. For any doubts which are not clear from this document, fell free to contact me at `marceloperlin@gmail.com`.

References

Carol Alexander. *Market Risk Analysis: Practical Financial Econometrics*. Wiley, 2008.

Chris Brooks. *Introduction to Econometrics*. Cambridge University Press, 2002.

James Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57 (2):357–84, March 1989. URL `http://ideas.repec.org/a/ecm/emetrp/v57y1989i2p357-84.html`.

James Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

James Hamilton. Regime switching models. *Palgrave Dictionary of Economics*, 2005.

J. Kim and R. Nelson. *State Space Model with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications*. The MIT Press, 1999.

Ruey Tsay. *Analysis of Financial Time Series*. John Wiley and Sons, 2002.

Peijie Wang. *Financial Econometrics*. Taylor and Francis, 2003.

Ding Zhuanxin. An implementation of markov regime switching model with time varying transition probabilities in matlab. *SSRN elibrary*, 2012. URL `http://ssrn.com/abstract=2083332`.