Dissertação de Mestrado

Inferência sobre os hiperparâmetros dos modelos estruturais usando *bootstrap*

por: **Juliana Aparecida Ribeiro**

Orientadora: **Glaura da Conceição Franco**

Co-orientador: **Frederico R. B. Cruz**

Fevereiro de 2006

JULIANA APARECIDA RIBEIRO

INFERÊNCIA SOBRE OS HIPERPARÂMETROS DOS MODELOS ESTRUTURAIS USANDO *BOOTSTRAP*

Dissertação apresentada ao Departamento de Estatística do Instituto de Ciências Exatas da UFMG como requisito parcial para obtenção do título de Mestre em Estatística.

Orientadora: Glaura da Conceição Franco
Co-orientador: Frederico R. B. Cruz

UNIVERSIDADE FEDERAL DE MINAS GERAIS

BELO HORIZONTE, FEVEREIRO DE 2006

Carinhosamente, dedico esta dissertação a meus pais, por terem confiado em mim e me apoiado em mais esta etapa de minha vida.

AGRADECIMENTOS

Primeiramente agradeço a Deus, que me iluminou, me deu forças, determinação e perseverança para alcançar mais esta conquista em minha vida.

Aos meus pais, grandes encorajadores e intercessores em mais esta caminhada. Quantas foram as vezes, que mesmo distantes fisicamente, estiveram ao meu lado, dizendo para eu não desistir, pois eu seria capaz de vencer os obstáculos. Foi difícil, mas com tamanho apoio e confiança dedicados, não teria como falhar.

Imensa gratidão a minha orientadora Profª. Glaura, pois sem ela, nada teria conseguido. Aproximadamente cinco anos de uma excelente convivência e impecável acompanhamento e orientação acadêmica. Obrigada por toda a amizade, paciência, apoio, confiança, conselhos, zelo e carinho.

Ao Prof. Frederico, que foi mais que um co-orientador. Esteve sempre presente, dispondo-se a me ensinar, se dedicando e acompanhando fielmente o desenvolvimento desta dissertação. Mediante incomensurável ajuda, fica a minha inestimável gratidão.

Agradeço ao Pedro, por ter permanecido junto a mim ao longo desta jornada. O carinho, a paciência e o bem-querer foram essenciais em todos os momentos.

Aos meus irmãos, parentes, amigos (as), colegas e todos, que de alguma forma, se fizeram presentes e me apoiaram, o meu eterno agradecimento!

RESUMO

Esta dissertação é baseada na decomposição de séries temporais via componentes não-observáveis, através dos chamados modelos estruturais. Uma forma alternativa de reescrever os modelos estruturais se dá por meio da forma de espaço de estados. Realizada esta transcrição, utiliza-se o chamado filtro de Kalman para a atualização do vetor de estado e para a construção da função de verossimilhança, tornando possível a estimação dos hiperparâmetros do modelo. Aplicações da técnica de reamostragem *bootstrap* são realizadas a fim de fazer inferências sobre os hiperparâmetros dos modelos, atentando-se, inclusive, para a construção de intervalos de confiança. Para tanto, são utilizadas implementações na linguagem de programação Ox. Resultados de simulações asseguram a eficiência do processo de estimação da linguagem Ox, juntamente à aplicação em uma série temporal real.

Palavras-chave: modelos estruturais, filtro de Kalman, hiperparâmetros, *bootstrap*.

ABSTRACT

This dissertation is based on the decomposition of times series via non-observed components, through structural models. An alternative way to rewrite the structural models is by using the state space form. Once this transcription is done, the Kalman filter is used for updating the state vector and constructing the likelihood function to estimate the hyperparameters of the model. The bootstrap resampling technique is applied to make inferences on the hyperparameters of the models, attending for the construction of confidence intervals, which is made in the programming language Ox. The results of the simulations and a real time series application verify the efficiency of the language estimation process.

Keywords: structural models, Kalman filter, hyperparameters, bootstrap.

Lista de Figuras

2.1. Um algoritmo quase-Newton.....15

2.2. Série simulada seguindo o MNL com $\sigma_{\eta}^{2} = 0{,}50$ e $\sigma_{\varepsilon}^{2} = 1{,}00$ 16

2.3. Gráficos das médias das estimativas do MNL de acordo com o *burn-in* 18

2.4. Gráficos das médias das estimativas do MNL de acordo com o MC 18

2.5. Gráficos das estimativas do MNL de acordo com o tamanho da série 19

2.6. Série simulada seguindo o MTL, com $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$ 20

2.7. Gráficos das médias das estimativas do MTL de acordo com o *burn-in* 22

2.8. Gráficos das médias das estimativas do MTL de acordo com o MC 23

2.9. Gráficos das estimativas do MTL de acordo com o tamanho da série 24

2.10. Série seguindo o MEB, com $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}01$, $\sigma_{\omega}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$ 25

2.11. Gráficos das médias das estimativas do MEB de acordo com o *burn-in* 27

2.12. Gráficos das médias das estimativas do MEB de acordo com o MC 28

2.13. Gráficos das estimativas do MEB de acordo com o tamanho da série 29

3.1. Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MNL 37

3.2. Histogramas das estimativas *bootstrap* do MNL de uma simulação MC 37

3.3. Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MTL 40

3.4. Histogramas das estimativas *bootstrap* do MTL de uma simulação MC 41

3.5. Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MEB 44

3.6. Histogramas das estimativas *bootstrap* do MEB de uma simulação MC 45

4.1. Série temporal do IPCA (%) de Belo Horizonte (jan/97 a out/05) 48

4.2. Histograma da série do IPCA (%) de Belo Horizonte 49

Lista de Tabelas

2.1. Resultado das simulações para MNL variando o tamanho da série e do *burn-in* e o número de replicações MC.....17

2.2. Resultado das simulações para MTL variando o tamanho da série e do *burn-in* e o número de replicações MC.....21

2.3. Resultado das simulações para MEB variando o tamanho da série e do *burn-in* e o número de replicações MC.....26

3.1. EMV e estimativas *bootstrap* dos hiperparâmetros do MNL 36

3.2. Intervalos de confiança *bootstrap* para os hiperparâmetros do MNL 38

3.3. EMV e estimativas *bootstrap* dos hiperparâmetros do MTL 39

3.4. Intervalos de confiança *bootstrap* para os hiperparâmetros do MTL 42

3.5. EMV e estimativas *bootstrap* dos hiperparâmetros do MEB 43

3.6. Intervalos de confiança *bootstrap* para os hiperparâmetros do MEB 46

4.1. Análise descritiva da série do IPCA (%) de Belo Horizonte 49

4.2. EMV e estimativas *bootstrap* dos hiperparâmetros do MEB para a série do IPCA (%) 50

4.3. EMV e estimativas *bootstrap* dos hiperparâmetros do MTL para a série do IPCA (%).....50

4.4. EMV e estimativas *bootstrap* dos hiperparâmetros do MNL para a série do IPCA (%).....51

B.1. Resultado das simulações para MNL variando o número de iterações BFGS e de replicações MC (*burn-in* = 100) 67

B.2. Resultado das simulações para MTL variando o número de iterações BFGS e de replicações MC (*burn-in* = 100) 67

B.3. Resultado das simulações para MEB variando o número de iterações BFGS e de replicações MC (*burn-in* = 100) 68

Lista de Abreviaturas e Símbolos

Y_t - Série temporal univariada $Y_1, Y_2, \ldots$

μ_t - Componente de tendência

γ_t - Componente sazonal

ψ_t - Componente cíclico

ε_t - Componente aleatório ou erro

MNL – Modelo de Nível Local

MTL – Modelo de Tendência Linear Local

MEB – Modelo Estrutural Básico

i.i.d. – Independente e identicamente distribuído

S - Número de períodos sazonais

FK – Filtro de Kalman

MC – Monte Carlo

N – Número de observações da série

Burn-in – Número de observações iniciais que são excluídas das séries simuladas

BFGS – Número máximo de iterações BFGS para convergência

EMV – Estimativas de máxima verossimilhança

EQM – Erro quadrático médio

Vício (%) – Vício percentual das estimativas

* – Indicativo dos resultados obtidos utilizando-se o *bootstrap*

SUMÁRIO

1. INTRODUÇÃO 1

1.1. Motivação 2

1.2. Objetivos e Escopo da Dissertação 2

1.3. Organização da Dissertação 3

2. MODELOS ESTRUTURAIS 4

2.1. Definição. 4

2.2. Tipos de Modelos..... 5

2.2.1. Modelo de Nível Local... 5

2.2.2. Modelo de Tendência Linear Local... 5

2.2.3. Modelo Estrutural Básico. 6

2.3. Forma de Espaço de Estados 7

2.3.1. Forma de espaço de estados para o MNL... 8

2.3.2. Forma de espaço de estados para o MTL... 9

2.3.3. Forma de espaço de estados para o MEB... 9

2.4. Filtro de Kalman.. 10

2.5. Estimação por Máxima Verossimilhança..... 12

2.6. Implementações em Ox..... 15

2.6.1. Monte Carlo para o MNL. 16

2.6.2. Monte Carlo para o MTL. 19

2.6.3. Monte Carlo para o MEB. 24

2.7. Conclusões e Observações Finais..... 29

3. *BOOTSTRAP*..... 30

3.1. Definição 30

3.2. A Técnica do *Bootstrap* 30

3.2.1. *Bootstrap* Paramétrico..... 30

3.2.2. *Bootstrap* Não-Paramétrico..... 31

3.3. Inferência Usando o *Bootstrap* 31

3.4. *Bootstrap* em Modelos Estruturais 32

3.5. Intervalos de Confiança *Bootstrap* Percentílico 34

3.6. Simulações do *Bootstrap* usando Ox 35

3.6.1. *Bootstrap* para o MNL. 36

3.6.2. *Bootstrap* para o MTL. 39

3.6.3. *Bootstrap* para o MEB. 43

3.7. Conclusões e Observações Finais..... 47

4. Aplicação a Série Real..... 48

5. Conclusões Finais..... 52

Apêndice A 54

Apêndice B 67

Apêndice C 69

Apêndice D 72

Referências Bibliográficas..... 73

1. INTRODUÇÃO

Uma série temporal é um conjunto de observações tomadas em tempos determinados, comumente em intervalos iguais. Exemplos de séries temporais são o consumo mensal de energia em uma residência, o preço semanal de um produto, o valor anual de um índice de produção industrial, o valor diário de fechamento de uma determinada ação nas bolsas de valores, as temperaturas horárias anunciadas pelo serviço meteorológico de uma cidade, e muitos outros.

Todos os métodos estatísticos de previsão de séries temporais baseiam-se na idéia de que as observações passadas da série contêm informações sobre o seu padrão de comportamento no futuro. A essência destes métodos consiste em identificar o padrão da série, separando-o do ruído (erro) contido nas observações individuais, e utilizá-lo para prever os valores futuros da série.

Matematicamente, uma série temporal é definida pelos valores $Y_1, Y_2, \ldots$ de uma variável Y nos tempos $t_1, t_2, \ldots$. Portanto, Y é uma função de t simbolizada por $Y = \mathrm{F}(t)$ e é representada ilustrativamente por meio da construção de um gráfico de Y em função de t, descrito por pontos que se movem com o decorrer do tempo.

Uma maneira de se modelar séries temporais é através da decomposição em seus componentes não-observáveis, por meio dos modelos estruturais (Harvey, 1989). Esta técnica de desmembrar as séries através dos componentes não-observáveis foi inicialmente menosprezada devido à dificuldade computacional envolvida na implementação do filtro de Kalman. Hoje em dia, os modelos estruturais vêm sendo bastante utilizados, tendo em vista que os procedimentos de modelagem e previsão das séries tornam-se mais simples devido à interpretação direta conseguida através dos componentes destes modelos. Além disto, pacotes computacionais, como o Ox (Doornik,1999), foram desenvolvidos para o ajuste de modelos estruturais e possibilitam a implementação de novas rotinas através da linguagem C++. Alguns exemplos de utilização desta modelagem podem ser vistos em Harvey (2001), Harvey *et al.* (2004) e Durbin & Koopman (2001).

A estimação do modelo é feita através da estimação das variâncias dos erros dos componentes não-observáveis, denominadas hiperparâmetros. Inferências sobre estas

quantidades podem ser realizadas a partir da distribuição assintótica dos hiperparâmetros (Harvey, 1989).

A técnica de reamostragem *bootstrap* (Efron, 1979) é um recurso alternativo para fazer inferência sobre os hiperparâmetros quando não se tem uma amostra grande. Não se trata de um método de estimação, e sim de uma técnica em que amostras dos dados originais são coletadas e a partir destas "reamostras" as estimações são realizadas. Alguns trabalhos utilizando o *bootstrap* em modelos estruturais já foram realizados, por exemplo: Stoffer & Wall (1991), Franco & Souza (2002) e Pfeffermann & Tiller (2004). Stoffer & Wall (1991) propuseram a utilização do *bootstrap* em modelos de espaço de estados, Franco & Souza (2002) estudaram o *bootstrap* aplicado nos modelos mais simples, que apresentavam somente tendência e já Pfeffermann & Tiller (2004) utilizaram o *bootstrap* paramétrico e não-paramétrico para estimar os modelos estruturais. Outras referências importantes são Davis & Yam (2003) e Olsson (2005).

1.1. Motivação

Atualmente, poucos trabalhos aplicando o *bootstrap* em modelos estruturais são conhecidos. Desta forma, a motivação principal desta dissertação baseia-se na utilização da metodologia *bootstrap* nos modelos estruturais. Inferência sobre os hiperparâmetros destes modelos, baseada na distribuição assintótica, pode ser problemática e pode gerar erros de estimação quando se tem poucas observações na série analisada. Uma forma, então, de tentar fazer inferência sobre os hiperparâmetros dos modelos estruturais se dá através da aplicação da técnica não-paramétrica do *bootstrap*, .

1.2. Objetivos e Escopo da Dissertação

O principal interesse desta dissertação é o estudo dos hiperparâmetros dos modelos estruturais em séries temporais, mais especificamente do modelo de nível local (MNL), do modelo de tendência linear local (MTL) e do modelo estrutural básico (MEB).

A dissertação restringir-se-á ao estudo sobre as variâncias dos ruídos de cada modelo, cuja distribuição será assumida como sendo a gaussiana. Para fazer inferências sobre tais hiperparâmetros, a técnica do *bootstrap*, introduzida por Efron (1979), será

utilizada, através da qual serão construídos intervalos de confiança por meio da linguagem Ox (Doornik, 1999).

Assim, pretende-se neste trabalho:

(i) fazer um estudo detalhado sobre os modelos estruturais em séries temporais, desmembrando-os na forma de espaço de estados e utilizando o filtro de Kalman (Kalman, 1960) para o cálculo da sua função de verossimilhança;

(ii) realizar simulações Monte Carlo para verificar a correção do método de estimação implementado na linguagem de programação matricial Ox e atestar a eficiência da implementação desenvolvida;

(iii) implementar o *bootstrap* nos resíduos e construir as séries *bootstrap* para os modelos especificados;

(iv) verificar a eficiência do *bootstrap* para construção da distribuição empírica dos hiperparâmetros e construção de intervalos de confiança;

(v) fazer aplicação em uma série temporal real.

1.3. Organização da Dissertação

No Capítulo 2 é feita uma revisão sobre os modelos estruturais e resultados de algumas simulações Monte Carlo são apresentados. O Capítulo 3 aborda a técnica não-paramétrica de reamostragem *bootstrap* e sua aplicação nos modelos estruturais. O Capítulo 4 contém aplicações da implementação em uma série temporal real. A conclusão do estudo, algumas observações e comentários finais encerram este trabalho, no Capítulo 5.

2. MODELOS ESTRUTURAIS

2.1. Definição

Para modelar e fazer previsões em uma série temporal é muito comum utilizar modelos de decomposição através de componentes não-observáveis, também denominados de modelos estruturais. Este tipo de modelagem supõe que os movimentos característicos de uma série temporal univariada possam ser decompostos em quatro componentes:

(i) Tendência (μ_t): refere-se à direção geral segundo a qual os dados temporais se desenvolvem ao longo de um intervalo de tempo.

(ii) Componente sazonal (γ_t): refere-se aos padrões repetitivos da série de interesse, com alguns períodos de elevação ou redução, que se reproduzem identicamente durante uma mesma "estação" (mês, semana, dia, etc.).

(iii) Componente cíclica (ψ_t): refere-se às oscilações que se repetem identicamente em longo prazo ou aos desvios em torno da reta ou da curva de tendência.

(iv) Componentes aleatórios ou erros (ε_t): referem-se aos deslocamentos esporádicos das séries temporais, provocados por eventos casuais.

Então, através do uso dos modelos estruturais, uma série temporal univariada Y_t, $t = 1,2,\ldots,n$, pode ser escrita através da expressão:

$$Y_t = \mu_t + \gamma_t + \psi_t + \varepsilon_t, \qquad t = 1,2,\ldots,n, \tag{1}$$

sendo $\varepsilon_t \overset{iid}{\sim} N(0,\sigma_\varepsilon^2)$.

2.2. Tipos de Modelos

Os tipos de modelos estruturais que serão abordados neste trabalho são o modelo de nível local (MNL), o modelo de tendência linear local (MTL) e o modelo estrutural básico (MEB), como pode ser melhor estudado em Harvey (1989).

2.2.1. Modelo de Nível Local

O modelo de nível local (MNL), também conhecido como modelo de passeio aleatório adicionado de um ruído, é o modelo mais simples, pois não há inclinação positiva ou negativa. Isto é, a série se move aleatoriamente sem seguir uma trajetória fixa. Este modelo contém um componente de nível (aleatório) e um erro e está definido por:

$$y_t = \mu_t + \varepsilon_t, \qquad \varepsilon_t \overset{iid}{\sim} N(0,\sigma_\varepsilon^2), \qquad t = 1,2,\ldots,n, \tag{2.a}$$

$$\mu_t = \mu_{t-1} + \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,\sigma_\eta^2), \tag{2.b}$$

supondo ε_t e η_t não-correlacionados.

2.2.2. Modelo de Tendência Linear Local

O modelo da tendência, quando a série tem um movimento crescente ou decrescente e também tem um comportamento de passeio aleatório, atende pelo nome de modelo de tendência linear local (MTL). Sabendo-se que $t = 1,2,\ldots,n$, tem-se:

$$y_t = \mu_t + \varepsilon_t, \qquad \varepsilon_t \overset{iid}{\sim} N(0,\sigma_\varepsilon^2), \tag{3.a}$$

$$\mu_t = \mu_{t-1} + \beta_{t-1} + \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,\sigma_\eta^2) \tag{3.b}$$

$$\beta_t = \beta_{t-1} + \xi_t, \qquad \xi_t \overset{iid}{\sim} N(0,\sigma_\xi^2), \tag{3.c}$$

sendo ε_t, η_t e ξ_t distúrbios tipo ruído branco mutuamente não-correlacionados.

2.2.3. Modelo Estrutural Básico

O modelo estrutural básico (MEB) é freqüentemente usado para modelar séries que contenham o componente sazonal. Ele pode ser definido por:

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \qquad \varepsilon_t \overset{iid}{\sim} N(0,\sigma_\varepsilon^2), \tag{4.a}$$

$$\mu_t = \mu_{t-1} + \beta_{t-1} + \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,\sigma_\eta^2) \tag{4.b}$$

$$\beta_t = \beta_{t-1} + \xi_t, \qquad \xi_t \overset{iid}{\sim} N(0,\sigma_\xi^2), \tag{4.c}$$

$$\gamma_t = -\gamma_{t-1} - \ldots - \gamma_{t-s+1} + \omega_t, \qquad \omega_t \overset{iid}{\sim} N(0,\sigma_\omega^2), \tag{4.d}$$

assumindo que $t = 1,2,\ldots,n$ e que s é o número de períodos sazonais, sendo ε_t, η_t, ξ_t e ω_t distúrbios tipo ruído branco mutuamente não-correlacionados, em que ω_t é a perturbação aleatória associada à sazonalidade no instante t.

O componente sazonal γ_t pode ser modelado através da modelagem sazonal por fatores. Para modelar um componente sazonal por fatores, é necessário impor a restrição de que a soma dos componentes sazonais seja zero, isto é,

$$\sum_{j=0}^{s-1} \gamma_{t-j} = 0. \tag{5.a}$$

Obtém-se uma modelagem estocástica para o componente sazonal fazendo

$$\sum_{j=0}^{s-1} \gamma_{t-j} = \omega_t, \tag{5.b}$$

sendo $\omega_t \overset{iid}{\sim} N(0,\sigma_\omega^2)$,

2.3. Forma de Espaço de Estados

Na Seção 2.2, foram descritos os modelos estruturais para uma série temporal univariada, cujos componentes não-observáveis são definidos através de 2 equações (MNL), 3 equações (MTL) e 4 equações distintas (MEB).

Sabe-se que para realizar a estimação de tais modelos é preciso encontrar as estimativas das variâncias dos ruídos relacionados a cada componente não-observável, os hiperparâmetros. A forma de espaço de estados foi uma maneira encontrada para se reescrever as expressões da modelagem estrutural, reduzindo-se o número de equações, fato tal que tornará o processo de estimação dos hiperparâmetros um pouco mais simples.

Na forma de espaço de estados é possível representar a modelagem estrutural através de duas equações: a equação das observações (ou de medida) e a equação do estado (ou transição), dadas respectivamente por:

$$y_t = \mathbf{z}_t'\alpha_t + d_t + \varepsilon_t, \qquad \varepsilon_t \sim N(0, h_t), \tag{6.a}$$

$$\alpha_t = \mathbf{T}_t\alpha_{t-1} + c_t + \mathbf{R}_t\eta_t, \qquad \eta_t \sim N(0, \mathbf{Q}_t), \tag{6.b}$$

sendo $t = 1,2,\ldots,n$. ε_t são ruídos não-correlacionados, η_t é um vetor de ruídos serialmente não-correlacionados cuja matriz de covariância é dada por $\mathbf{Q}_t$ e α_t é o vetor de estado. Sabe-se também que ε_t e η_t são independentes entre si, $\mathbf{z}_t, d_t, \mathbf{T}_t, \mathbf{c}_t$ e $\mathbf{R}_t$ são chamados matrizes do sistema e são conhecidas após a definição da modelagem a ser aplicada à série. Nesta dissertação, assumir-se-á $d_t = 0$ e $c_t = 0$, indicando que serão modeladas séries univariadas sem a presença de outras séries influentes nesta modelagem.

O modelo na forma de espaço de estados estabelece a independência do futuro do processo em relação ao passado, dado o estado presente. O estado do processo condensa todas as informações do passado necessárias para a predição do futuro. Devem-se atender às seguintes suposições:

(i) $E(\alpha_0) = \mathbf{a}_0$ e $Cov(\alpha_0, \alpha_0') = \mathbf{P}_0$, sendo α_0 o vetor de estado inicial.

(ii) $E(\varepsilon_t' \eta_z) = 0$ e $E(\varepsilon_t' \alpha_0) = E(\eta_t \alpha_0') = 0$, $\forall z,\ t = 1,2,\ldots,n$

Atendendo a estas suposições, qualquer modelo para uma série temporal pode ser escrito como uma combinação linear dos ruídos ε_t e η_t presentes e passados e do vetor de estado inicial α_0. Para aqueles parâmetros desconhecidos contidos nos componentes da forma de espaço de estados, denota-se um vetor ψ que é conhecido como o vetor de hiperparâmetros. Neste trabalho, estes hiperparâmetros serão as variâncias presentes no modelo e que poderão ser estimadas pelo processo de máxima verossimilhança.

2.3.1. Forma de espaço de estados para o MNL

Como foi visto na Seção 2.2, o modelo de nível local é o modelo mais simples, pois não segue nenhuma tendência fixa e é composto apenas por um componente de nível adicionado de um ruído. A forma de espaço de estados pode ser aplicada ao MNL, encontrando-se os seguintes valores de suas variáveis:

$$z_t' = 1, \qquad d_t = 0, \qquad \varepsilon_t = \varepsilon_t, \qquad T_t = 1, \qquad c_t = 0, \qquad R_t = 1,$$

$$\alpha_t = \mu_t, \qquad \alpha_{t-1} = \mu_{t-1}, \qquad h_t = \sigma_\varepsilon^2, \qquad Q_t = \sigma_\eta^2, \qquad \eta_t = \eta_t.$$

Então, o MNL pode ser escrito através das expressões:

$$y_t = \alpha_t + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \sigma_\varepsilon^2), \qquad t = 1,2,\ldots,n, \tag{7.a}$$

$$\alpha_t = \alpha_{t-1} + \eta_t, \qquad \eta_t \sim N(0, \sigma_\eta^2)\ . \tag{7.b}$$

2.3.2. Forma de espaço de estados para o MTL

Assumindo que a série temporal tenha um movimento crescente ou decrescente e também um comportamento de passeio aleatório, serão obtidas as seguintes matrizes de sistema:

$$\mathbf{z}_t' = [1 \quad 0], \qquad c_t = d_t = 0, \qquad R_t = 1,$$

$$\mathbf{T}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \qquad h_t = \sigma_\varepsilon^2, \qquad \mathbf{Q}_t = \begin{bmatrix} \sigma_\eta^2 & 0 \\ 0 & \sigma_\xi^2 \end{bmatrix}.$$

O vetor de estado é dado por $\alpha_t = \begin{bmatrix} \mu_t \\ \beta_t \end{bmatrix}$ e, por conseguinte, $\alpha_{t-1} = \begin{bmatrix} \mu_{t-1} \\ \beta_{t-1} \end{bmatrix}$, sendo o vetor de resíduos $\eta_t = \begin{bmatrix} \eta_t \\ \xi_t \end{bmatrix}$. Desta forma, tem-se:

$$y_t = [1 \quad 0] \begin{bmatrix} \mu_t \\ \beta_t \end{bmatrix} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \sigma_\varepsilon^2), \tag{8.a}$$

$$\begin{bmatrix} \mu_t \\ \beta_t \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_{t-1} \\ \beta_{t-1} \end{bmatrix} + \begin{bmatrix} \eta_t \\ \xi_t \end{bmatrix}, \qquad \eta_t \sim N(0, \mathbf{Q}_t),$$

$$\mathbf{Q}_t = Cov(\eta_t) = \begin{bmatrix} \sigma_\eta^2 & 0 \\ 0 & \sigma_\xi^2 \end{bmatrix}. \tag{8.b}$$

2.3.3. Forma de espaço de estados para o MEB

Para os modelos com sazonalidade modelados como visto na Seção 2.2.3, têm-se as seguintes matrizes de sistema:

$$\mathrm{z}_t' = \begin{bmatrix} 1 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}_{1\times(s+1)}, \qquad c_t = d_t = 0, \qquad R_t = 1,$$

$$\mathbf{T}_t = \begin{bmatrix} 1 & 1 & | & & & 0 & & \\ 0 & 1 & | & & & \sim & & \\ - & - & - & - & - & - & - & - \\ & & | & -1 & -1 & \dots & -1 & -1 \\ & & | & 1 & 0 & \dots & 0 & 0 \\ 0 & & | & 0 & 1 & \dots & 0 & 0 \\ \sim & & | & \vdots & \vdots & \ddots & \vdots & \vdots \\ & & | & 0 & 0 & \dots & 1 & 0 \end{bmatrix}_{(s+1)\times(s+1)},$$

$$h_t = \sigma_\varepsilon^2, \qquad \mathbf{Q}_t = \begin{bmatrix} \sigma_\eta^2 & & & & & \\ & \sigma_\xi^2 & & & 0 & \\ & & \sigma_\omega^2 & & \sim & \\ & & & 0 & & \\ & 0 & & & \ddots & \\ & \sim & & & & 0 \end{bmatrix}_{(s+1)\times(s+1)}.$$

A forma de espaço de estados pode então ser composta pelo vetor de estado

$$\alpha_t = \begin{bmatrix} \mu_t \\ \beta_t \\ \gamma_t \\ \gamma_{t-1} \\ \vdots \\ \gamma_{t-s+2} \end{bmatrix}_{(s+1)\times 1},$$

com vetor de resíduos $\eta_t = [\eta_t \quad \xi_t \quad \omega_t \quad 0 \quad \dots \quad 0]'_{(s+1)\times 1}$ e sabendo-se que s representa o número de períodos sazonais.

2.4. Filtro de Kalman

Para fazer previsões em um modelo na forma de espaço de estados a partir de observações passadas até o tempo t, é necessário que o vetor de estado α_t seja atualizado e, para isso, deve-se sempre atualizar os estimadores deste componente não-observável através do chamado filtro de Kalman (1960).

O filtro de Kalman decompõe a série através de equações recursivas que atualizam seqüencialmente o vetor de estado. Suas características básicas são atualização, previsão e suavização.

Sabendo-se que Y_{t-1} é o vetor das observações até o instante $t-1$, $E(\alpha_0) = \mathbf{a}_0$ e $Cov(\alpha_0, \alpha_0') = \mathbf{P}_0$, tem-se que:

$$\text{(i) } (\alpha_t \mid Y_{t-1}) \sim N(\mathbf{a}_{t|t-1}, \mathbf{P}_{t|t-1}), \text{ sendo } \mathbf{a}_{t|t-1} = E(\alpha_t \mid Y_{t-1}) = \mathbf{T}_t \mathbf{a}_{t-1} + c_t, \tag{9.a}$$

$$\text{(ii) } \mathbf{P}_{t|t-1} = Var(\alpha_t \mid Y_{t-1}) = \mathbf{T}_t \mathbf{P}_{t-1} \mathbf{T}_t' + \mathbf{R}_t \mathbf{Q}_t \mathbf{R}_t', \tag{9.b}$$

$$\text{(iii) } (y_t \mid Y_{t-1}) \sim N(\tilde{y}_{t|t-1}, F_t), \text{ sendo } \tilde{y}_{t|t-1} = E(y_t \mid Y_{t-1}) = \mathbf{z}_t' \mathbf{a}_{t|t-1} + d_t, \tag{9.c}$$

$$\text{(iv) } F_t = Var(y_t \mid Y_{t-1}) = \mathbf{z}_t' \mathbf{P}_{t|t-1} \mathbf{z}_t + h_t. \tag{9.d}$$

Então, pode-se notar que a média e a matriz de covariância do vetor de estado no tempo t são respectivamente iguais a $\mathbf{a}_{t+1|t}$ e $\mathrm{P}_{t+1|t}$. Através de recursividade, encontram-se os seguintes resultados importantes:

$$\mathbf{a}_{t+1|t} = \mathbf{a}_t = \mathbf{a}_{t|t-1} + \mathbf{P}_{t|t-1} . \mathbf{z}_t . F_t^{-1} . (y_t - \tilde{y}_{t|t-1}), \tag{10.a}$$

$$\mathbf{P}_{t+1|t} = \mathbf{P}_t = \mathbf{P}_{t|t-1} - \mathbf{P}_{t|t-1} . \mathbf{z}_t . F_t^{-1} . \mathbf{z}_t' . \mathbf{P}_{t|t-1}. \tag{10.b}$$

As equações (10.a) e (10.b) são chamadas de equações de atualização, pois através delas é possível atualizar os estimadores $\mathbf{a}_t$ e $\mathbf{P}_t$ do vetor de estado no instante t. Para facilitar o manuseio das fórmulas, chama-se de v_t o erro de previsão ou inovações, sendo que $E(v_t) = 0$ e $Var(v_t) = F_t$, e chama-se de $\mathbf{K}_t$ a matriz de ganho. Assim, tem-se que:

$$v_t = y_t - \tilde{y}_{t|t-1} \Rightarrow v_t = y_t - \mathbf{z}_t'\mathbf{a}_{t|t-1} - d_t, \tag{11.a}$$

$$\mathbf{K}_t = \mathbf{T}_{t+1}.\mathbf{P}_{t|t-1}.\mathbf{z}_t.F_t^{-1}. \tag{11.b}$$

Então, substituindo v_t e o $\mathbf{K}_t$ em (10.a) e (10.b) é possível listar as equações para o filtro de Kalman (9.d), (10.a), (10.b), (11.a) e (11.b).

Um exemplo é mostrado aplicando o filtro de Kalman no MNL. Tomando-se as equações (7.a) e (7.b) na forma de espaço de estados do MNL e os valores encontrados das matrizes de sistema, substituindo-os nas fórmulas do filtro de Kalman e atendendo às suposições e exposições vistas anteriormente, encontram-se para o MNL equações mais simples e fáceis de ser usadas, porém com a mesma finalidade:

$$v_t = y_t - \mathbf{a}_{t|t-1}, \tag{12.a}$$

$$\mathbf{a}_{t+1|t} = \mathbf{a}_{t|t-1} + \mathbf{K}_t.v_t, \tag{12.b}$$

$$F_t = \mathbf{P}_{t|t-1} + \sigma_\varepsilon^2, \tag{12.c}$$

$$\mathbf{K}_t = \mathbf{P}_{t|t-1} / F_t, \tag{12.d}$$

$$\mathbf{P}_{t+1|t} = \mathbf{P}_{t|t-1} - \mathbf{K}_t^2.F_t + \sigma_\eta^2. \tag{12.e}$$

Para a inicialização do filtro de Kalman toma-se $\mathbf{a}_0 = 0$ ou $\mathbf{a}_0 = E(Y_t)$ e $\mathbf{P}_0$ suficientemente grande (Harvey, 2001).

2.5. Estimação por Máxima Verossimilhança

Como mencionado nas seções anteriores, a estimação dos modelos estruturais ocorre através da estimação das variâncias dos ruídos relacionados aos componentes não-observáveis, que são conhecidas como hiperparâmetros. Costumeiramente, referem-se aos hiperparâmetros por meio de um vetor simbolizado pela letra grega *psi* (ψ). No caso do MNL, $\psi = (\sigma_\eta^2, \sigma_\varepsilon^2)$. Para o MTL, $\psi = (\sigma_\eta^2, \sigma_\xi^2, \sigma_\varepsilon^2)$ e no caso do MEB, tem-se que $\psi = (\sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, \sigma_\varepsilon^2)$.

A estimação por máxima verossimilhança pode ser usada para se estimar os hiperparâmetros de um modelo através da maximização da função densidade conjunta $f(y_1, \ldots, y_n; \psi)$ em relação a ψ, que é o vetor de hiperparâmetros. Para se fazer esta estimação, deve-se calcular a função de verossimilhança, que será o produtório das distribuições preditivas $L(\psi) = f(y_1, \ldots, y_n \mid \psi) = \prod_{t=1}^{n} f(y_t \mid Y_{t-1}, \psi)$, e assim achar o valor de ψ que a maximiza.

Usando-se o modelo na forma de espaço de estados, a função de verossimilhança é obtida através do erro de previsão v_t do filtro de Kalman, supondo que $(y_t \mid Y_{t-1}) \sim N(\tilde{y}_{t|t-1}, F_t)$. Então, a função de verossimilhança será descrita por:

$$L(\psi) = \prod_{t=1}^{n} (2\pi)^{-1/2} \mid F_t \mid^{-1/2} \exp\left\{ -\frac{1}{2} \left(y_t - \tilde{y}_{t|t-1}\right)' F_t^{-1} \left(y_t - \tilde{y}_{t|t-1}\right) \right\}. \qquad (13)$$

Substituindo $v_t = y_t - \tilde{y}_{t|t-1}$ tem-se:

$$L(\psi) = \prod_{t=1}^{n} (2\pi)^{-1/2} \mid F_t \mid^{-1/2} \exp\left\{ -\frac{1}{2} v_t' F_t^{-1} v_t \right\}. \qquad (14)$$

Aplicando-se o logaritmo natural para simplificar os cálculos:

$$\log L(\psi) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\log|F_t| - \frac{1}{2}\sum_{t=1}^{n} v_t' F_t^{-1} v_t \,. \qquad (15)$$

Para estimar o vetor ψ de hiperparâmetros, deve-se então encontrar um valor para a incógnita tal que maximize a verossimilhança. Pelo fato de a função de verossimilhança ser uma função complicada do vetor de hiperparâmetros, esta estimação não será realizada analiticamente, mas via método numérico, implementado em ambiente Ox (Doornik 1999).

A discussão sobre métodos para solucionar problemas de otimização pode alongar-se indefinidamente. Assim, preferiu-se neste trabalho apenas anunciar que o método de maximização a ser utilizado é o conhecido BFGS (em referência aos seus autores, Broyden-Fletcher-Goldfarb-Shanno), pelo seu desempenho satisfatório no passado, conforme visto, por exemplo, em Franco (1998) e Franco & Souza (2002), entre outros. Certamente muitos outros métodos de maximização irrestrita são aplicáveis, havendo uma vasta literatura a ser consultada sobre esta e outras técnicas de otimização. Entre outros, veja Fletcher (1987), Gill et al. (1981), Cramer (1986) e o clássico livro de Press et al. (1988). Note que muitos textos sobre otimização abordam problemas de minimização em vez de maximização, mas, claro, isto é apenas o caso de reversão de um sinal algébrico.

O tratamento de detalhes profundos do procedimento BFGS está fora do escopo deste trabalho, mas em linhas gerais pode-se dizer que o método encontra-se no intermediário entre a simplicidade do método do gradiente e a rapidez do método de Newton, razão pela qual é conhecido como um método quase-Newton. No método BFGS, assim como em outros métodos quase-Newton, ao invés de a matriz hessiana ser calculada exatamente, como é o caso do método de Newton, ela é aproximada por um processo iterativo finito, via derivadas de primeira ordem, procurando um compromisso entre a rapidez de convergência e a dificuldade da avaliação da inversa a cada passo.

O método BFGS segue o algoritmo apresentado de forma bastante simplificada na Figura 2.1, mas que define claramente parâmetros importantes do algoritmo, tais como o número máximo de iterações, MAXIT, a tolerância máxima do vetor gradiente, e_1, e a tolerância máxima do passo, e_2.

algoritmo

{dados $f : R^n \rightarrow R$)

escolha $\boldsymbol{\psi}_0 \in R^n$ (por exemplo, $\psi_0 = 1$)

k←0

repita

calcule $\nabla f(\boldsymbol{\psi}^k)$ (gradiente)

calcule $D(\boldsymbol{\psi}^k)$ (aproximação para a inversa da hessiana $H^{-1}(\boldsymbol{\psi}^k)$)

$\boldsymbol{\psi}^{k+1} \leftarrow \boldsymbol{\psi}^k - \lambda_k D(\boldsymbol{\psi}^k)\nabla f(\boldsymbol{\psi}^k)$

k←k+1

até $(k \geq \text{MaxIt})$ **ou** $\left(\left|\nabla f(\psi^k)\right| \leq e_1\right)$ **ou** $\left(\left|\psi^k - \psi^{k-1}\right| \leq e_2\right)$

fim algoritmo

Figura 2.1: Um algoritmo quase-Newton

2.6. Implementações em Ox

Utilizando a linguagem Ox (Doornik, 1999), programou-se o ajuste dos três modelos estruturais apresentados na Seção 2.2, implementando-se rotinas que construíram a função de verossimilhança e a maximizava usando o método BFGS. Mediante a obtenção das estimativas de máxima verossimilhança dos hiperparâmetros, objetivou-se, principalmente, verificar a eficiência de tal implementação realizada.

A fim de conhecer e estabelecer fatores determinantes e influentes neste estudo Monte Carlo, foram avaliados o número de iterações necessárias para a convergência no algoritmo BFGS, o tamanho do *burn-in* para a simulação das séries (que corresponde ao número de observações excluídas no princípio da série simulada, que são fortemente influenciadas pelos valores iniciais assumidos), número de replicações Monte Carlo e o tamanho das séries simuladas. Optou-se por trabalhar com 3 tamanhos de séries: pequeno (*n=50*), médio (*n=100*) e grande (*n=200*).

Para tanto, serão analisadas as médias das estimativas de cada hiperparâmetro, assim como os seus respectivos erros quadráticos médios (EQM). Os EQM's serão calculados a partir da expressão:

$$EQM = \frac{\sum (\hat{\sigma}^2 - \sigma^2)^2}{MC}, \tag{16}$$

em que $\hat{\sigma}^2$ representa a estimativa de tal hiperparâmetro, σ^2 representa o valor real do hiperparâmetro e MC indica o número de replicações Monte Carlo.

De posse de valores predefinidos dos hiperparâmetros de cada modelo, construíram-se, então, rotinas computacionais (vide Apêndice A) que foram capazes de fornecer as médias e os EQM's das estimativas simuladas. Foram consideradas as precisões $\varepsilon_1=1.10^{-4}$ e $\varepsilon_2=5.10^{-3}$, para o algoritmo BFGS.

Em um estudo preliminar, variou-se o número máximo de iterações necessárias para a convergência do algoritmo BFGS (*BFGS=50, 200* e *1.000*) para cada um dos três modelos considerados. Adotando-se 4 casas decimais e analisando-se as estimativas, observou-se que não houve nenhuma mudança nas estimativas quando se aumentou o número máximo de iterações na implementação do método BFGS, sendo assim, optou-se por prosseguir o estudo assumindo *BFGS=50* (resultados apresentados no Apêndice B).

2.6.1. Monte Carlo para o MNL

Franco (1998) mostra, através de simulações, que os MNL's com $\sigma_\eta^2 \leq \sigma_\varepsilon^2$ caracterizam séries temporais com uma menor flutuação do nível μ_t em torno do valor médio das observações. Atentando-se para esta informação, ao longo deste trabalho, serão analisados os MNL's cujos hiperparâmetros são definidos como sendo $\sigma_\eta^2 = 0{,}50$ e $\sigma_\varepsilon^2 = 1{,}00$. Uma destas séries pode ser vista na Figura 2.2.

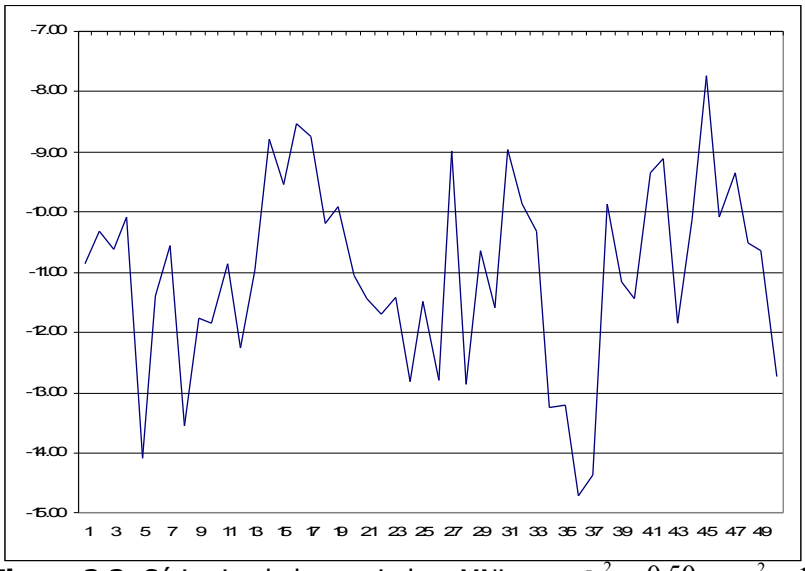

Figura 2.2: Série simulada seguindo o MNL com $\sigma_\eta^2 = 0{,}50$ e $\sigma_\varepsilon^2 = 1{,}00$

Variando o tamanho da série, o número de replicações MC, o tamanho do *burn-in* e assumindo *BFGS = 50*, têm-se os resultados apresentados na Tabela 2.1.

Tabela 2.1: Resultado das simulações para MNL variando o tamanho da série, do *burn-in* e o número de replicações MC

Tamanho da série	*Burn-in*	Nº de Replicações MC	$\sigma_{\eta}^{2} = 0{,}50$		$\sigma_{\varepsilon}^{2} = 1{,}00$	
			Média	EQM	Média	EQM
50	100	100	0,4872	0,0833	1,0109	0,1185
		500	0,5109	0,0868	0,9911	0,1082
		1.000	0,5133	0,0873	0,9871	0,1065
	500	100	0,4968	0,0795	0,9393	0,1062
		500	0,4927	0,0794	0,9983	0,1054
		1.000	0,5029	0,0827	0,9965	0,1095
100	100	100	0,4845	0,0477	1,0033	0,0627
		500	0,4985	0,0435	0,9958	0,0560
		1.000	0,4985	0,0400	0,9953	0,0541
	500	100	0,4918	0,0432	0,9703	0,0520
		500	0,4988	0,0397	0,9979	0,0523
		1.000	0,4965	0,0378	1,0051	0,0506
200	100	100	0,4994	0,0217	0,9868	0,0233
		500	0,5029	0,0197	0,9923	0,0237
		1.000	0,5019	0,0185	0,9950	0,0244
	500	100	0,5072	0,0182	0,9920	0,0289
		500	0,4926	0,0179	0,9938	0,0256
		1.000	0,4988	0,0188	0,9966	0,0270

Realizando-se análises gráficas das estimativas da Tabela 2.1 de acordo com o tamanho do *burn-in*, o número de replicações MC e o tamanho da série, obteve-se:

(i) Tamanho do *burn-in*: foram calculadas as médias das estimativas cujo *burn-in=100* e daquelas cujo *burn-in=500*.

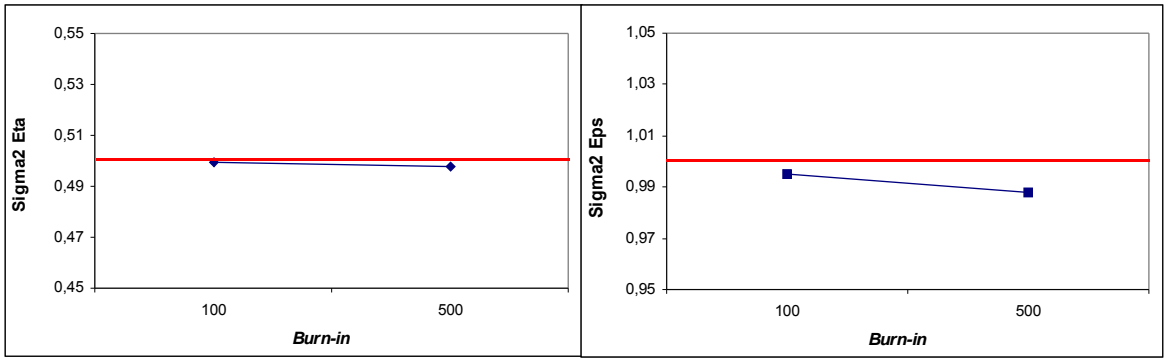

Figura 2.3: Gráficos das médias das estimativas do MNL de acordo com o *burn-in*

Através da Figura 2.3 conclui-se que as médias das estimativas obtidas adotando o tamanho do *burn-in=100* estão mais próximas dos valores reais assumidos para os hiperparâmetros (indicados pelas linhas horizontais nos gráficos) que as estimativas obtidas utilizando *burn-in=500*.

(ii) Número de replicações MC: adotando-se *burn-in=100*, foram calculadas as médias das estimativas obtidas de acordo com *MC=100, 500 e 1.000*.

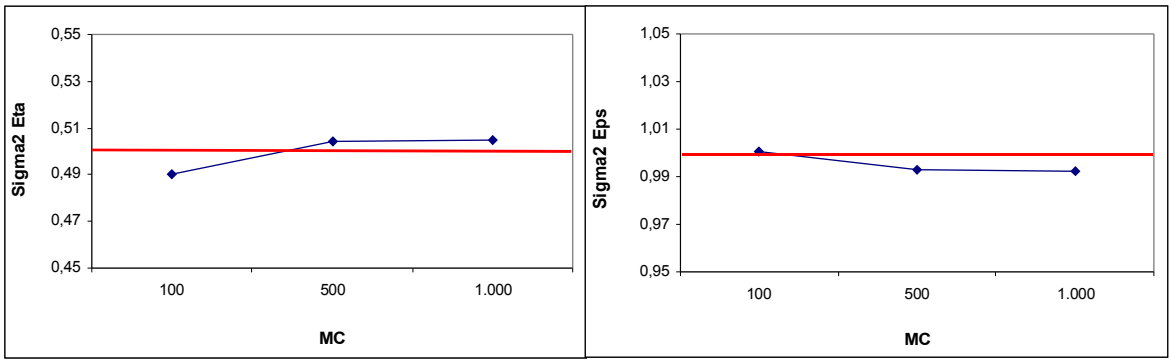

Figura 2.4: Gráficos das médias das estimativas do MNL de acordo com o MC

Nota-se que há uma tendência à estabilização das médias das estimativas de ambos os hiperparâmetros à medida que o número de repetições Monte Carlo aumenta. Verifica-se também que a diferença entre as médias das estimativas com *MC=500* e *MC=1.000* está na quarta casa decimal, optando-se assim pela adoção de *MC=500*, a fim de reduzir o tempo computacional.

1. Tamanho da série: adotando-se *burn-in=100* e *MC=500*, foram analisadas as estimativas obtidas de acordo com *n=50, 100 e 200*.

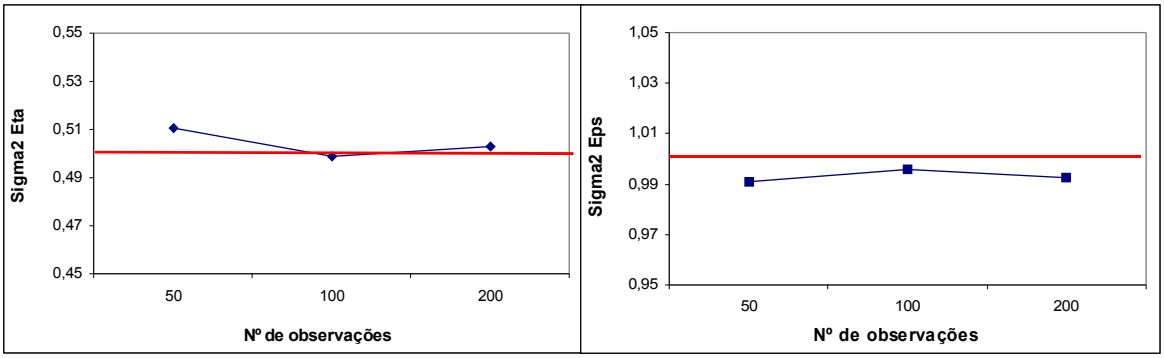

Figura 2.5: Gráficos das estimativas do MNL de acordo com o tamanho da série

Ambos os gráficos da Figura 2.5 mostram que todas as estimativas se aproximam muito dos valores reais assumidos para o MNL, adotando-se *n=50, 100* e *200*. Percebe-se também que a diferença entre as estimativas é muito pequena, encontrando-se apenas na terceira casa decimal (exceto a diferença entre as estimativas do hiperparâmetro σ_{η}^{2}, referentes a *n=50* e *n=100*, que se encontra na segunda casa decimal).

2.6.2. Monte Carlo para o MTL

Adotou-se um MTL com hiperparâmetros definidos como sendo $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$. Uma destas séries pode ser vista na Figura 2.6.

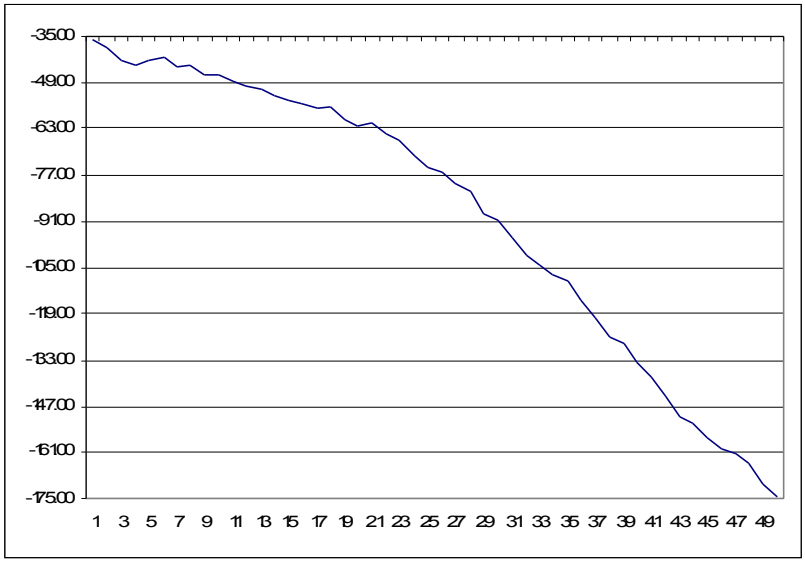

Figura 2.6: Série simulada seguindo o MTL, com $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$

Na Figura 2.6 tem-se o gráfico de uma das séries simuladas com os hiperparâmetros $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$, sendo nítida a tendência decrescente dos dados temporais.

Variando o tamanho da série, o número de replicações MC, o tamanho do *burn-in* e assumindo *BFGS = 50*, têm-se os resultados apresentados na Tabela 2.2.

Tabela 2.2: Resultado das simulações para MTL variando o tamanho da série, do *burn-in* e o número de replicações MC

Tamanho da série	*Burn-in*	Nº de Replicações MC	$\sigma_{\eta}^{2} = 0{,}50$		$\sigma_{\xi}^{2} = 0{,}10$		$\sigma_{\varepsilon}^{2} = 1{,}00$	
			Média	EQM	Média	Média	EQM	Média
50	100	100	0,5344	0,3318	0,0883	0,0051	0,9534	0,1621
		500	0,5506	0,3763	0,0951	0,0060	0,9763	0,1517
		1.000	0,5701	0,3930	0,0933	0,0057	0,9648	0,1546
	500	100	0,2103	0,3047	0,1117	0,0058	2,3231	4,4501
		500	0,2598	0,3790	0,1028	0,0052	2,1768	3,8230
		1.000	0,2758	0,4002	0,1004	0,0050	2,2202	4,0250
100	100	100	0,5416	0,2388	0,0932	0,0042	0,9399	0,1120
		500	0,5420	0,2239	0,0926	0,0031	0,9748	0,0952
		1.000	0,5381	0,2379	0,0949	0,0031	0,9851	0,0890
	500	100	0,4311	0,3822	0,0989	0,0030	1,4127	0,7398
		500	0,2953	0,3239	0,1051	0,0031	1,6192	1,1078
		1.000	0,2971	0,3118	0,1061	0,0031	1,6418	1,1787
200	100	100	0,5895	0,1301	0,0952	0,0018	0,9542	0,0483
		500	0,5276	0,1217	0,1002	0,0017	0,9834	0,0479
		1.000	0,5039	0,1191	0,1003	0,0017	0,9950	0,0459
	500	100	0,3617	0,2601	0,1093	0,0020	1,3435	0,5746
		500	0,3197	0,2209	0,1076	0,0017	1,3610	0,4569
		1.000	0,3211	0,2304	0,1085	0,0017	1,3624	0,4507

Realizando-se análises gráficas das estimativas da Tabela 2.2 de acordo com o tamanho do *burn-in*, o número de replicações MC e o tamanho da série, obteve-se:

(i) Tamanho do *burn-in*: foram calculadas as médias das estimativas cujo *burn-in=100* e daquelas cujo *burn-in=500*.

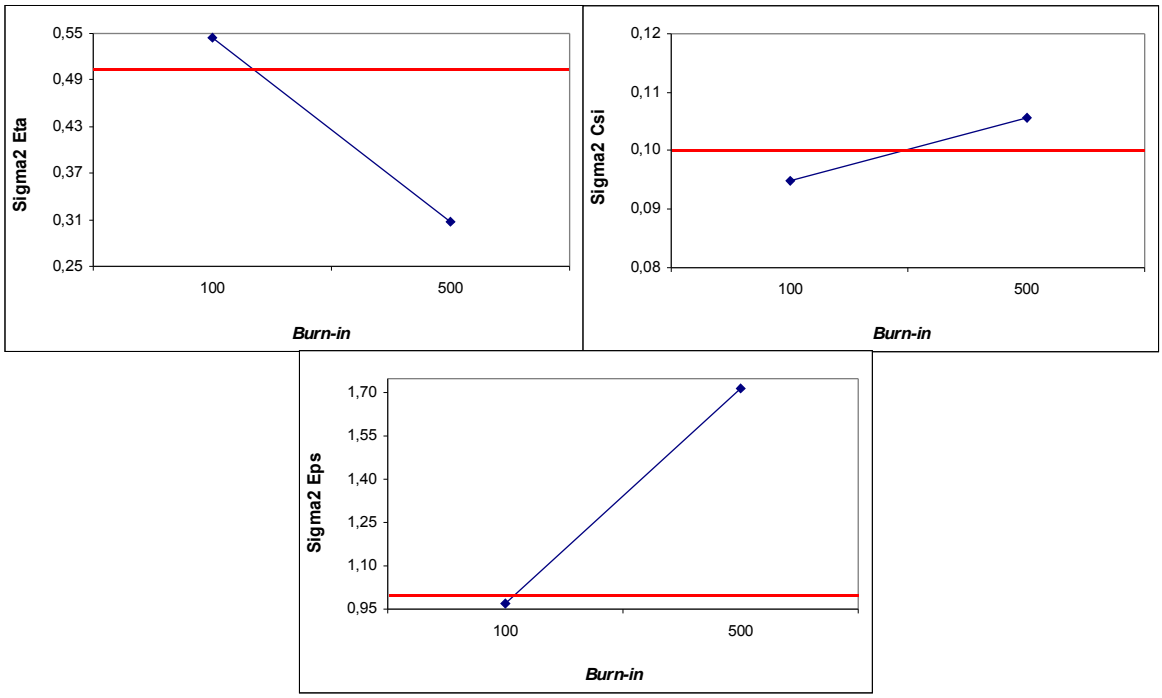

Figura 2.7: Gráficos das médias das estimativas do MTL de acordo com o *burn-in*

Através da Figura 2.7 conclui-se que as médias das estimativas obtidas adotando o tamanho do *burn-in=100* estão mais próximas dos valores reais assumidos para os hiperparâmetros (indicados pelas linhas horizontais nos gráficos) que as estimativas obtidas utilizando *burn-in=500*.

(ii) Número de replicações MC: adotando-se *burn-in=100*, foram calculadas as médias das estimativas obtidas de acordo com *MC=100, 500 e 1.000*.

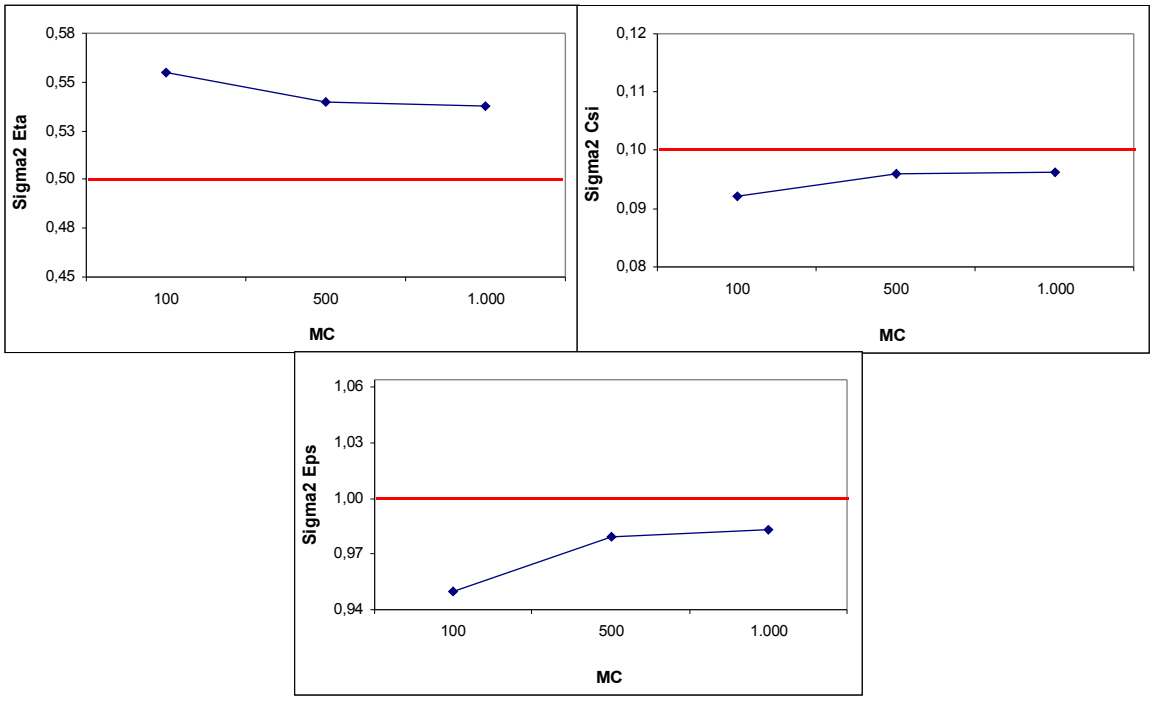

Figura 2.8: Gráficos das médias das estimativas do MTL de acordo com o MC

Os gráficos da Figura 2.8 mostram que, adotando-se *MC=1.000*, as médias das estimativas dos hiperparâmetros são mais próximas dos valores reais assumidos. Verifica-se, também, que houve a estabilização na estimação dos hiperparâmetros, pois as estimativas obtidas adotando-se *MC=500* são muito próximas daquelas obtidas quando se adotou *MC=1.000*. Sendo assim, por economia de tempo computacional, prefere-se adotar *MC=500*.

(iii) Tamanho da série: adotando-se *burn-in=100* e *MC=500*, foram analisadas as estimativas obtidas de acordo com *n=50, 100 e 200*.

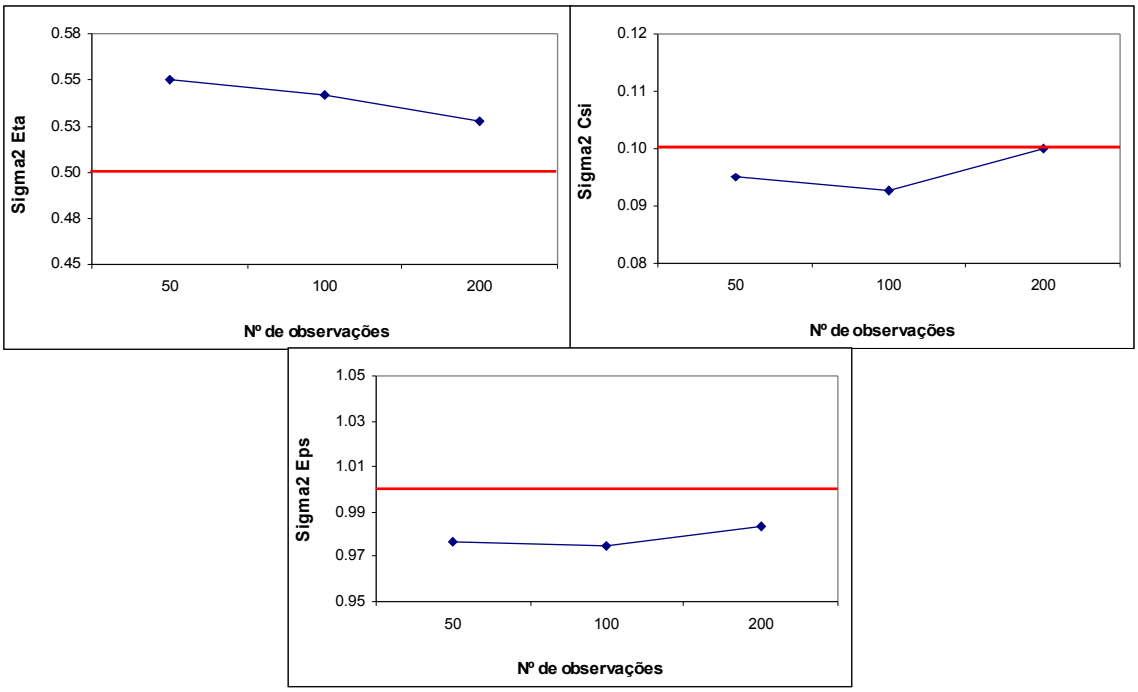

Figura 2.9: Gráficos das estimativas do MTL de acordo com o tamanho da série

Os gráficos da Figura 2.9 mostram que as estimativas que mais se aproximam dos valores reais assumidos são aquelas obtidas nas simulações contendo 200 observações. Percebe-se também, que à medida que se aumenta o número de observações simuladas, em geral diminui-se o tamanho do vício obtido nas estimativas.

2.6.3. Monte Carlo para o MEB

Será assumido um MEB com sazonalidade trimestral (*s=4)* e com hiperparâmetros definidos como sendo $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}01$, $\sigma_{\omega}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$. Uma destas séries pode ser vista na Figura 2.10.

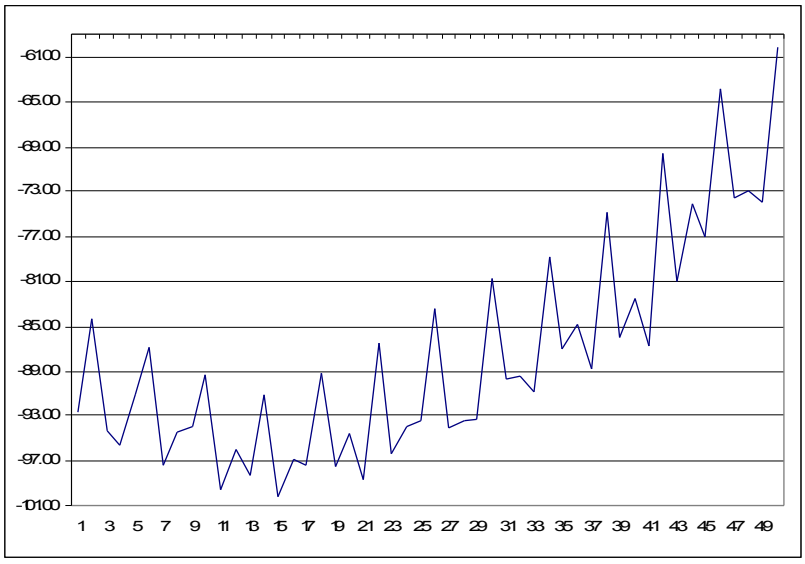

Figura 2.10: Série seguindo o MEB, com $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}01$, $\sigma_{\omega}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$

Na Figura 2.10 tem-se o gráfico de uma das séries simuladas do MEB com os hiperparâmetros $\sigma_{\eta}^{2} = 0{,}50$, $\sigma_{\xi}^{2} = 0{,}01$, $\sigma_{\omega}^{2} = 0{,}10$ e $\sigma_{\varepsilon}^{2} = 1{,}00$, sendo fácil perceber a presença sazonal e também a não-estacionariedade dos dados em torno de uma média, ou seja, a série apresenta tendências crescentes e decrescentes.

Variando o tamanho da série, o número de replicações MC, o tamanho do *burn-in* e assumindo *BFGS = 50*, têm-se os resultados apresentados na Tabela 2.3.

Tabela 2.3: Resultado das simulações para MEB variando o tamanho da série, do *burn-in* e o número de replicações MC

Tamanho da série	*Burn-in*	Nº de Replicações MC	$\sigma_\eta^2 = 0{,}50$		$\sigma_\xi^2 = 0{,}01$		$\sigma_\omega^2 = 0{,}10$		$\sigma_\varepsilon^2 = 1{,}00$	
			Média	EQM	Média	EQM	Média	EQM	Média	EQM
50	100	100	0,4094	0,1560	0,0137	0,0004	0,0910	0,0093	0,9941	0,1615
		500	0,4541	0,1601	0,0113	0,0003	0,1001	0,0078	0,9936	0,2186
		1.000	0,4649	0,1635	0,0114	0,0003	0,1030	0,0089	0,9860	0,2286
	500	100	0,4439	0,2263	0,0144	0,0004	0,0845	0,0078	1,1611	0,4216
		500	0,4115	0,2362	0,0143	0,0004	0,0899	0,0077	1,2126	0,4066
		1.000	0,4299	0,2311	0,0136	0,0003	0,0924	0,0081	1,1837	0,4179
100	100	100	0,5173	0,1120	0,0105	0,0001	0,0996	0,0028	1,0151	0,1430
		500	0,4859	0,0877	0,0108	0,0001	0,1015	0,0035	0,9934	0,1203
		1.000	0,4849	0,0853	0,0104	0,0001	0,1008	0,0035	1,0052	0,1151
	500	100	0,5015	0,1419	0,0118	0,0002	0,0980	0,0034	1,0939	0,2033
		500	0,4460	0,1193	0,0108	0,0001	0,0925	0,0035	1,1398	0,2134
		1.000	0,4562	0,1291	0,0110	0,0001	0,0937	0,0034	1,1217	0,2039
200	100	100	0,5072	0,0417	0,0094	0,0000	0,1021	0,0016	0,9728	0,0479
		500	0,4941	0,0435	0,0097	0,0000	0,1000	0,0015	1,0004	0,0568
		1.000	0,4919	0,0481	0,0100	0,0000	0,1004	0,0015	1,0054	0,0584
	500	100	0,5186	0,0697	0,0104	0,0001	0,0984	0,0014	1,0399	0,0824
		500	0,4871	0,0555	0,0106	0,0000	0,0969	0,0016	1,0576	0,0866
		1.000	0,4790	0,0536	0,0106	0,0000	0,0964	0,0016	1,0595	0,0854

Realizando-se análises gráficas das estimativas da Tabela 2.3 de acordo com o tamanho do *burn-in*, o número de replicações MC e o tamanho da série, obteve-se:

(i) Tamanho do *burn-in*: foram calculadas as médias das estimativas cujo *burn-in=100* e daquelas cujo *burn-in=500*.

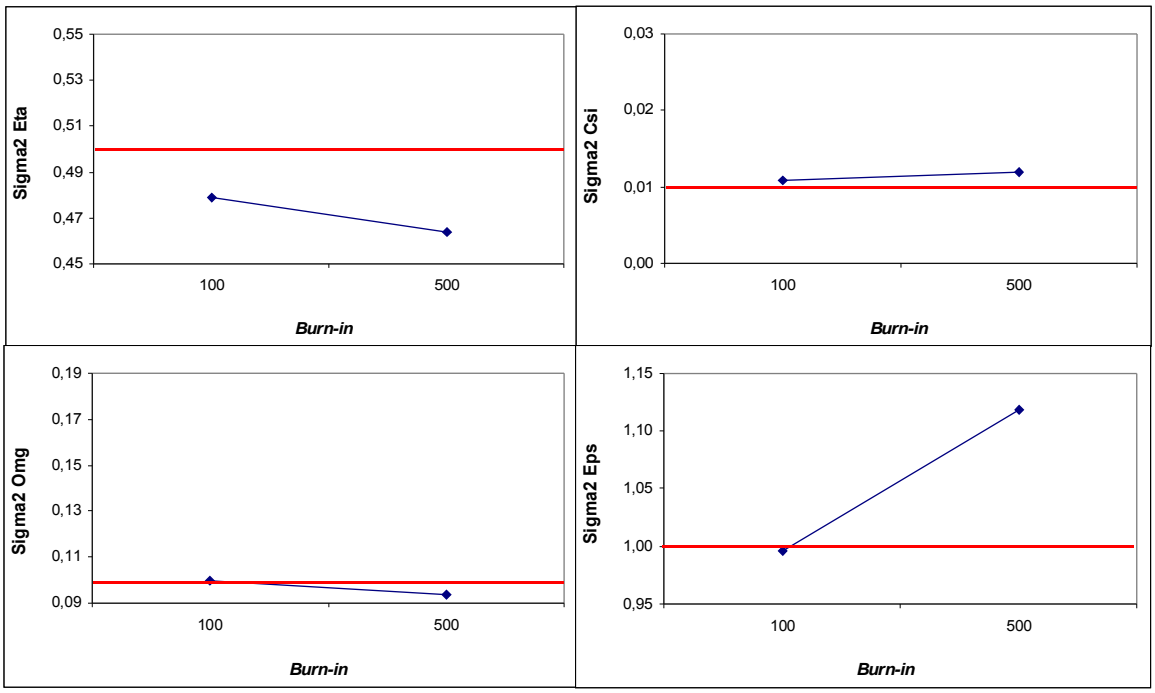

Figura 2.11: Gráficos das médias das estimativas do MEB de acordo com o *burn-in*

Através da Figura 2.11 conclui-se que as médias das estimativas obtidas adotando o tamanho do *burn-in=100* estão mais próximas dos valores reais assumidos para os hiperparâmetros (indicados pelas linhas horizontais nos gráficos).

(ii) Número de replicações MC: adotando-se *burn-in=100*, foram calculadas as médias das estimativas obtidas de acordo com *MC=100, 500 e 1.000*.

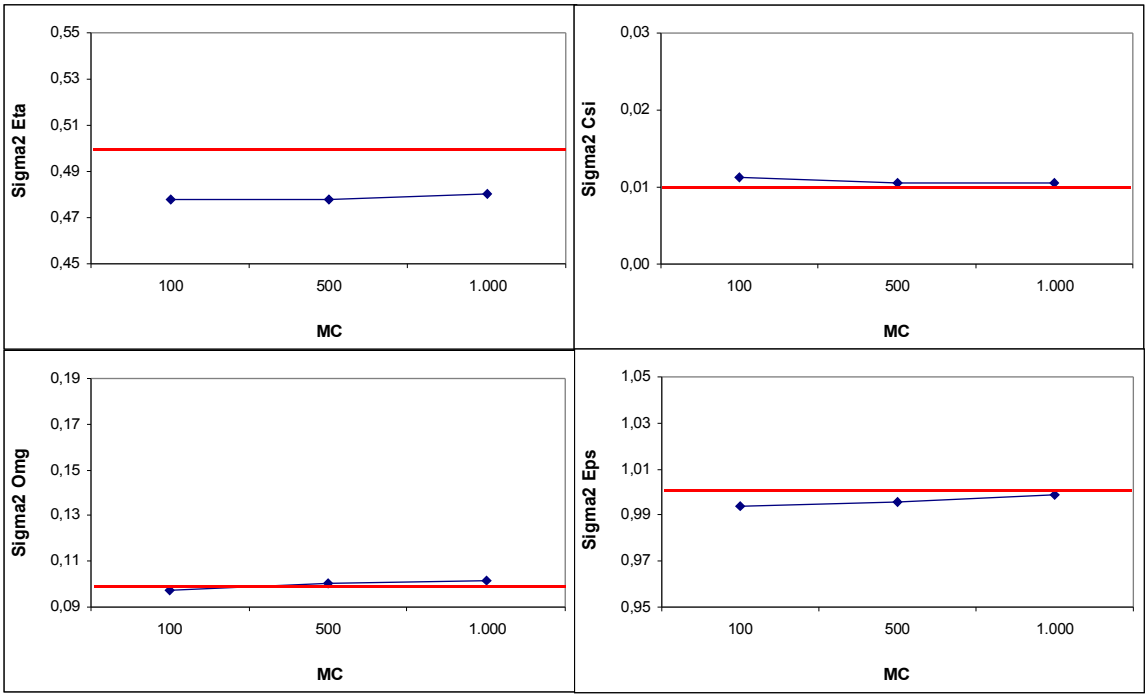

Figura 2.12: Gráficos das médias das estimativas do MEB de acordo com o MC

A Figura 2.12 mostra que as médias das estimativas de σ_{η}^{2}, σ_{ξ}^{2} e σ_{ε}^{2} ficam mais próximas dos valores reais assumidos quando é adotado *MC=1.000*. Já a estimativa de σ_{ω}^{2} fica mais próxima do valor real deste hiperparâmetro quando adota-se *MC=500*. Entretanto, nos resultados dos quatro hiperparâmetros, percebe-se que as diferenças entre as estimativas adotando-se *MC=500* e aquelas obtidas adotando-se *MC=1.000* são muito pequenas. Visualiza-se também, que as estimativas praticamente se estabilizaram quando se adotou *MC=500*, tornando-se, então, desnecessário utilizar *MC=1.000*.

(iii) Tamanho da série: adotando-se *burn-in=100* e *MC=500*, foram analisadas as estimativas obtidas de acordo com *n=50, 100 e 200*.

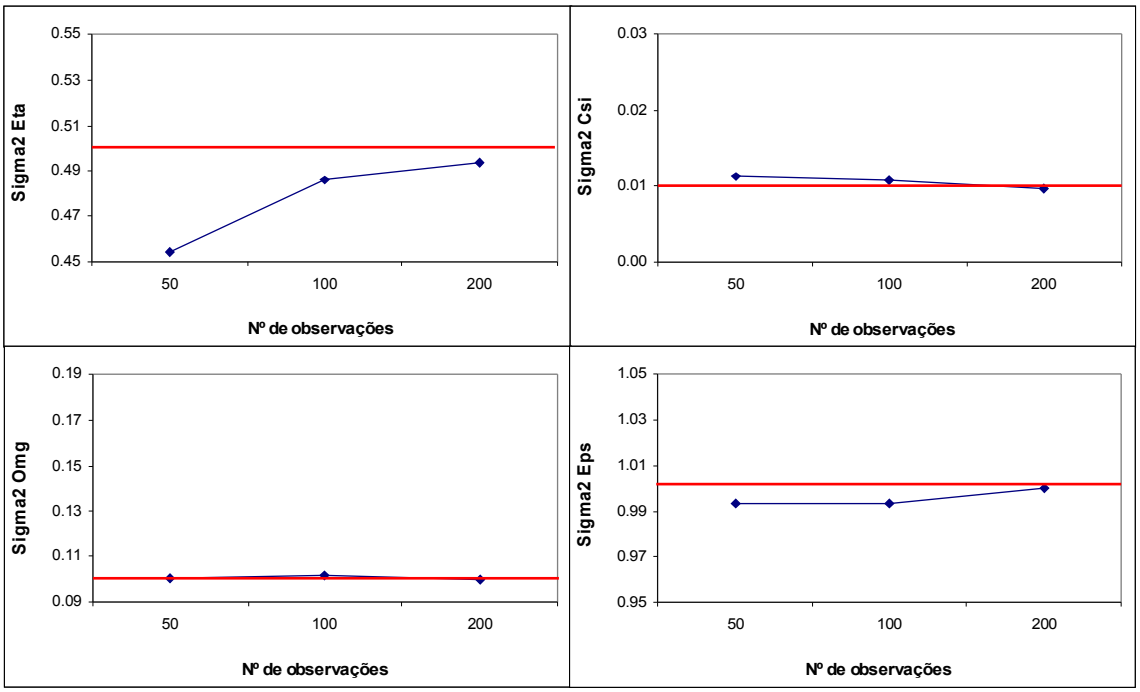

Figura 2.13: Gráficos das estimativas do MEB de acordo com o tamanho da série

A Figura 2.13 mostra que as médias das estimativas de todos os hiperparâmetros ficam mais próximas dos valores reais assumidos quando é adotado *n=200*. Percebe-se então, que à medida que se aumenta o número de observações simuladas, diminui-se o tamanho do vício obtido nas estimativas.

2.7. Conclusões e Observações Finais

Foi de grande valia a realização de estudos Monte Carlo neste momento do trabalho, para que alguns fatores influentes nas implementações pudessem ser definidos e para que fosse possível sentir quão precisas são as estimativas fornecidas pela implementação em Ox.

Através dos resultados obtidos foi possível concluir que o processo de estimação implementado em Ox foi satisfatório e eficiente. De um modo geral, para todos os modelos ajustados, foram observados baixos vícios nas estimativas dos valores esperados dos hiperparâmetros e pequenos erros quadráticos médios (EQM).

A princípio, foi possível notar, mediante os gráficos de algumas das séries simuladas, que as séries de cada um dos modelos, seguem realmente a trajetória esperada, ou seja, aleatoriedade num mesmo nível (MNL), tendência ao crescimento/decrescimento (MTL) e sazonalidade (MEB), assim como definido nas seções 2.2.1-2.2.3.

Conjecturando sobre o número máximo de iterações necessárias para a convergência do algoritmo BFGS, foi possível descobrir que o menor valor avaliado (*BFGS=50*) poderá ser assumido para a realização dos demais experimentos, tendo em vista que o aumento do mesmo não acarretou nenhuma mudança nas estimativas.

Avaliando o número de observações que devem ser retiradas do início de cada série simulada, devido a efeitos de valores iniciais assumidos, observou-se que não foi encontrada nenhuma melhoria entre as estimativas quando se aumentou este fator, podendo-se então empregar nas rotinas computacionais *burn-in=100*.

Sob o aspecto do número de replicações Monte Carlo, observou-se que sua variação (*MC=100, 500 e 1.000*) afetou, diferentemente, as estimativas de cada modelo. De modo geral, as estimativas obtidas adotando-se *MC=500* e *MC=1.000* foram bastante semelhantes e próximas dos valores reais assumidos, notando-se a estabilização da estimação já em *MC=500*. Deste modo, considerando a economia de tempo computacional gasto nas simulações, preferiu-se adotar em estudos seguintes *MC=500*.

Analisando os resultados obtidos com os diferentes tamanhos de séries (*n=50, 100 e 200*), verificou-se que na maioria dos experimentos em que se adotou *n=200*, os vícios das estimativas dos hiperparâmetros eram menores que nos demais casos. Assim sendo, pode-se concluir que os estimadores analisados são assintoticamente não-viesados.

3. *BOOTSTRAP*

3.1. Definição

O *bootstrap* é um método de reamostragem de dados introduzido por Efron (1979), usado especialmente na inferência estatística como, por exemplo, quando há interesse na construção de intervalos de confiança, testes de hipóteses, estimação de vícios, previsão, seleção de modelos e outros.

O *bootstrap* é um método em que são realizadas reamostragens (com reposição) dos dados de uma amostra de tamanho *n* finito, tentando aproximar a distribuição de uma determinada função das observações pela distribuição empírica dos dados.

3.2. A Técnica do *Bootstrap*

Existem basicamente duas maneiras de se realizar o *bootstrap*: paramétrica e não-paramétrica. Na primeira, reamostram-se observações da distribuição geradora dos dados e na última, a reamostragem é feita na própria amostra obtida. As duas formas serão descritas nas seções seguintes com maiores detalhes.

3.2.1. *Bootstrap* Paramétrico

No *bootstrap* paramétrico são feitas reamostragens (com reposição) de uma distribuição conhecida da qual os dados são obtidos. Neste caso, existem $X_1, X_2, \ldots, X_n$ variáveis aleatórias independentes com função de distribuição comum conhecida *F* e tem-se θ como um vetor de parâmetros desconhecidos. Então as amostragens serão realizadas diretamente em F_θ, conseguindo assim uma amostra *bootstrap* que, usando a notação de Efron & Tibshirani (1993), é denotada por $X^* = X_1^*, X_2^*, \ldots, X_n^*$.

3.2.2. *Bootstrap* Não-Paramétrico

Como o próprio nome diz, no *bootstrap* não-paramétrico não se assume uma distribuição para os dados. As amostragens são feitas a partir da amostra original de tamanho *n*, sendo necessária a suposição de que cada dado tenha identicamente uma massa de probabilidade igual a *1/n*, designada também, por uma suposta distribuição empírica $\hat{F}$. Neste caso, a amostra *bootstrap* também será feita com reposição, retirando-se um conjunto de *n* observações da amostra original e que, seguindo a notação de Efron & Tibshirani (1993), também será denotada por $X^* = X_1^*, X_2^*, \ldots, X_n^*$.

Como o *bootstrap* não-paramétrico não depende da distribuição que os dados seguem (distribuição desconhecida), o mesmo pode ser usado para qualquer conjunto de dados, tendo então maior aplicabilidade que o *bootstrap* paramétrico.

3.3. Inferência Usando o *Bootstrap*

Na prática, o tamanho mais utilizado para cada amostra *bootstrap* é o tamanho *n* da amostra original. Se o interesse está voltado para uma estatística *T* conjunta, dada por $T(X_1, X_2, \cdots, X_n)$, torna-se necessário gerar repetidas amostras *bootstrap* (paramétricas ou não-paramétricas) de tamanho *n*, denotadas por $X^{*1}, X^{*2}, \ldots, X^{*B}$, onde *B* é o número de reamostragens *bootstrap*.

A estimativa numa amostra *bootstrap* se aproxima do valor real quando $B \to \infty$, ou seja, quando as reamostragens são realizadas várias vezes, sendo que o valor de *B* é escolhido de acordo com a finalidade para a qual o método *bootstrap* está sendo usado.

Utilizando-se o princípio "*plug-in*", introduzido por Efron (1979), fazendo-se *B* reamostragens e calculando-se o valor da estatística de interesse $T^*(X^{*b})$, sendo que $b = 1,2,\ldots,B$, pode-se aproximar a distribuição empírica de $T^*(X^*)$ da verdadeira distribuição de $T(X)$ e então estimar o vetor de parâmetros desconhecidos θ.

3.4. *Bootstrap* em Modelos Estruturais

Lembrando que o objetivo deste estudo é a estimação do vetor de hiperparâmetros ψ de séries temporais utilizando modelos estruturais, agora serão utilizadas as técnicas de reamostragem do *bootstrap* não-paramétrico para auxiliar na estimação dos valores das variâncias desconhecidas dos ruídos dos modelos.

Sabe-se que para o MNL, $\psi = (\sigma_\eta^2, \sigma_\varepsilon^2)$, para o MTL, $\psi = (\sigma_\eta^2, \sigma_\xi^2, \sigma_\varepsilon^2)$ e para o MEB, $\psi = (\sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, \sigma_\varepsilon^2)$. Desta forma, o *bootstrap* será utilizado para reamostrar os dados originais, repetidas vezes e, a partir de tais reamostras, estimar por máxima verossimilhança os hiperparâmetros, conseguindo assim uma série de estimativas *bootstrap* para os mesmos.

Sabendo-se que para fazer inferências usando o *bootstrap* é necessário que a suposição de independência seja válida e levando-se em consideração que as observações das séries temporais são correlacionadas, deve-se fazer uma pequena modificação nos métodos descritos na Seção 3.2, aplicando o *bootstrap* nos resíduos, utilizando-se assim de um algoritmo que possibilita a aplicação do *bootstrap* em modelos estruturais, proposto por Stoffer & Wall (1991).

Através das recursividades do filtro de Kalman (Seção 2.4), serão obtidas as inovações ou erros de previsão ($v_t \sim N(0, F_t)$) e sua variância (F_t), que estarão em função de ψ e será usada a notação $v_t(\psi)$ e $F_t(\psi)$, respectivamente. A partir de então, deve-se estimar os hiperparâmetros através do método da máxima verossimilhança e em seguida, os erros de previsão estimados serão conseguidos, $v_t(\hat{\psi})$, $t = 1,2,\ldots,n$, e seu valor médio estimado como:

$$\bar{v}_t(\hat{\psi}) = \frac{\sum_{j=1}^{n} v_j(\hat{\psi})}{n}. \tag{17}$$

Padronizando as inovações, tem-se que

$$e_t(\hat{\psi}) = \frac{v_t(\hat{\psi}) - \bar{v}_t(\hat{\psi})}{\sqrt{F_t(\hat{\psi})}} . \qquad (18)$$

Reamostrando com reposição os $e_t(\hat{\psi})$, têm-se as inovações *bootstrap* $e_t^*(\hat{\psi})$ que serão usadas na construção da série *bootstrap*.

Utilizando as recursividades do filtro de Kalman, pode-se calcular $F_t(\psi)$ e $\mathbf{K}_t(\psi)$, e substituindo e_t por e_t^*, pode-se então construir a série *bootstrap* y_t^*, fazendo um processo inverso ao utilizado na estimação dos hiperparâmetros.

Recordando da Seção 2.4 e assumindo que serão utilizadas somente séries univariadas, ou seja, $c_t = d_t = 0$, sabe-se que:

$$\mathbf{a}_{t+1|t} = \mathbf{T}_t\mathbf{a}_{t|t-1} + \mathbf{K}_t v_t , \qquad (19)$$

$$v_t = y_t - \tilde{y}_{t|t-1} \;\Rightarrow\; y_t = v_t + \mathbf{z}_t'\mathbf{a}_{t|t-1} . \qquad (20)$$

Definindo o vetor $\mathbf{S}_t = \begin{bmatrix} \mathbf{a}_{t+1|t} \\ y_t \end{bmatrix}$, tem-se:

$$\mathbf{S}_t = \mathbf{A}_t\mathbf{S}_{t-1} + \mathbf{B}_t e_t, \qquad t = 1,2,\ldots,n , \qquad (21)$$

sendo $\mathbf{A}_t = \begin{bmatrix} \mathbf{T}_t & 0 \\ \mathbf{z}_t & 0 \end{bmatrix}$ e $\mathbf{B}_t = \begin{bmatrix} \mathbf{K}_t\sqrt{F_t} \\ \sqrt{F_t} \end{bmatrix}$.

De um modo geral, a nova série y_t^* poderá ser obtida resolvendo-se a equação (21), substituindo e_t por e_t^* e utilizando os valores estimados de $F_t(\hat{\psi})$ e $\mathbf{K}_t(\hat{\psi})$.

Obtida, então, a série *bootstrap* y_t^*, o filtro de Kalman será utilizado novamente e através do método de máxima verossimilhança será possível obter as estimativas *bootstrap* dos hiperparâmetros.

3.5. Intervalos de Confiança *Bootstrap* Percentílico

Os intervalos de confiança *bootstrap* percentílico são intervalos simples de serem obtidos através das replicações *bootstrap* de uma série original (Efron & Tibshirani, 1986).

A princípio são geradas *B* amostras *bootstrap* da série original, sendo que são estimados, para cada replicação, o vetor $\hat{\psi}^*$ de hiperparâmetros dos modelos, ordenando-se, em seguida, estas estimativas.

Assumindo que $\hat{G}^*$ seja a função de distribuição acumulada do estimador *bootstrap* $\hat{\psi}^*$, ou seja,

$$G^*(x) = P^*(\hat{\psi}^* \leq x),$$

tem-se que um intervalo *bootstrap* percentílico com *(1-2λ)%* de confiança será definido como os λ e *(1-λ)-ésimos* percentis de $\hat{G}^*$, da seguinte forma:

$$\left[G^{*\,-1}_{(\lambda)} \;;\; G^{*\,-1}_{(1-\lambda)} \right].$$

Pela definição tem-se que $G^{*\,-1}_{(\lambda)} = \hat{\psi}^*_{(\lambda)}$ é o *100.λ-ésimo* percentil empírico da distribuição *bootstrap*, ou seja, $\hat{\psi}^*_{(\lambda)}$ será o *B.λ-ésimo* valor das estimativas ordenadas obtidas através das replicações *bootstrap*.

Então, pode-se definir os limites inferior e superior de um intervalo de confiança *bootstrap* percentílico ao nível de *(1-2λ)*, como sendo:

$$\left[\hat{\psi}^{*\,(\lambda)}_b \;;\; \hat{\psi}^{*\,(1-\lambda)}_b \right].$$

3.5. **Simulações do *Bootstrap* usando Ox**

Na Seção 2.6 foram realizadas implementações com os objetivos de verificar quão precisas são as estimativas obtidas em Ox e de determinar os valores de alguns fatores importantes. Neste estudo preliminar, foram definidos que em simulações subseqüentes seriam adotados *BFGS=50*, *burn-in=100* e *MC=500*.

Tais valores definidos foram utilizados e juntamente aos valores predefinidos dos hiperparâmetros, rotinas computacionais na linguagem Ox foram então implementadas. Com o objetivo de obter estimativas *bootstrap* dos hiperparâmetros, seus EQM's, intervalos de 95% de confiança para as estimativas e as coberturas destes intervalos, foram realizadas *B=1.000* reamostragens *bootstrap*.

Para cada modelo, a princípio, estabeleceram-se os valores dos hiperparâmetros que seriam adotados. Com tais valores predefinidos foi gerada uma série através da qual estimaram-se seus hiperparâmetros e foram estimados os erros de previsão. A partir da série dos erros de previsão padronizados, realizaram-se *B=1.000* reamostragens e seguindo as explicações apresentadas na Seção 3.4, foram obtidas as séries *bootstrap*. Para cada uma destas novas séries obtiveram-se as estimativas *bootstrap*. Ao final de cada simulação, calculou-se a média destas *1.000* estimativas. Todo este processo foi repetido *MC=500* vezes, obtendo-se como resultado final, as médias das *500* médias das estimativas *bootstrap*.

A avaliação das estimativas fornecidas utilizando o Ox será realizada por meio de análises das médias das estimativas de cada hiperparâmetro e dos vícios percentuais das mesmas. Estes vícios serão calculados a partir da expressão:

$$Vício\ (\%) = \frac{\hat{\sigma}^2 - \sigma^2}{\sigma^2} \times 100\,. \qquad (22)$$

O erro quadrático médio (EQM) das estimativas, definido na Seção 2.6, também será analisado.

3.6.1. *Bootstrap* para o MNL

Através das rotinas computacionais apresentadas no Apêndice A, obteve-se os resultados apresentados na Tabela 3.1.

Tabela 3.1: EMV e estimativas *bootstrap* dos hiperparâmetros do MNL

Tamanho da série	Hiperparâmetro	EMV			Estimativas *Bootstrap*		
		Média	Vício (%)	EQM	Média*	Vício* (%)	EQM*
50	$\sigma_\eta^2 = 0{,}50$	0,4935	-1,3	0,0708	0,5545	10,9	0,0921
	$\sigma_\varepsilon^2 = 1{,}00$	0,9914	-0,9	0,1104	0,9553	-4,5	0,1206
100	$\sigma_\eta^2 = 0{,}50$	0,5091	1,8	0,0373	0,5347	6,9	0,0426
	$\sigma_\varepsilon^2 = 1{,}00$	0,9961	-0,4	0,0495	0,9815	-1,9	0,0533
200	$\sigma_\eta^2 = 0{,}50$	0,5008	0,2	0,0178	0,5125	2,5	0,0190
	$\sigma_\varepsilon^2 = 1{,}00$	1,0013	0,1	0,0259	0,9941	-0,6	0,0272

Através da Tabela 3.1, percebe-se que as estimativas *bootstrap* obtidas estão próximas das EMV's, em especial, no caso em que se utiliza *n=200*. Apesar desta proximidade entre os resultados, nota-se que os vícios percentuais das EMV's são sempre inferiores que os obtidos nas estimativas *bootstrap*. Outro fato importante refere-se à tendência decrescente do vício, à medida em que se aumenta o tamanho das séries simuladas, como se pode visualizar na Figura 3.1.

Ainda na Tabela 3.1, observa-se que nos três tamanhos de série analisados, os EQM's obtidos por máxima verossimilhança são menores que aqueles estimados quando se utiliza a técnica de reamostragem *bootstrap*. À medida que se aumenta o tamanho da série, diminuem os EQM's, tanto nas EMV's quanto nas estimativas *bootstrap*.

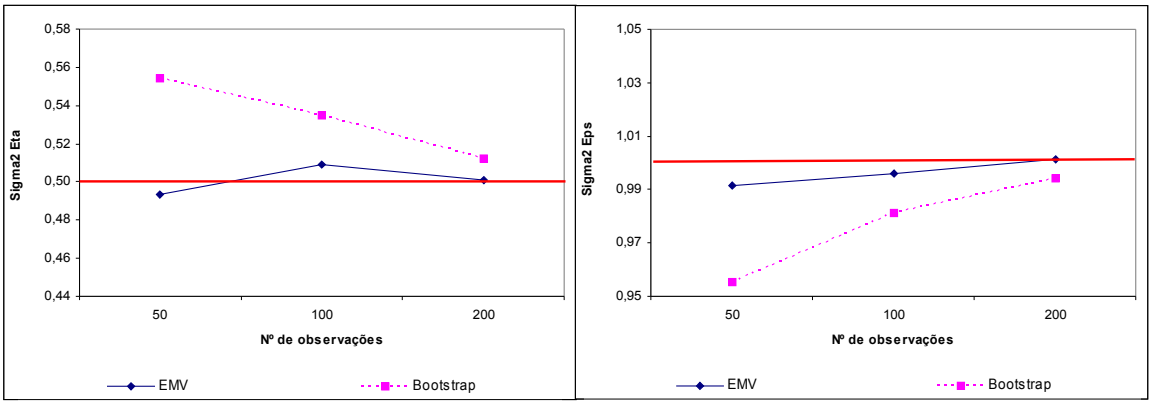

Figura 3.1: Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MNL

Tomando-se *1.000* estimativas *bootstrap* de uma das simulações, construíram-se histogramas a fim de analisar a forma da distribuição empírica das estimativas de cada hiperparâmetro do MNL.

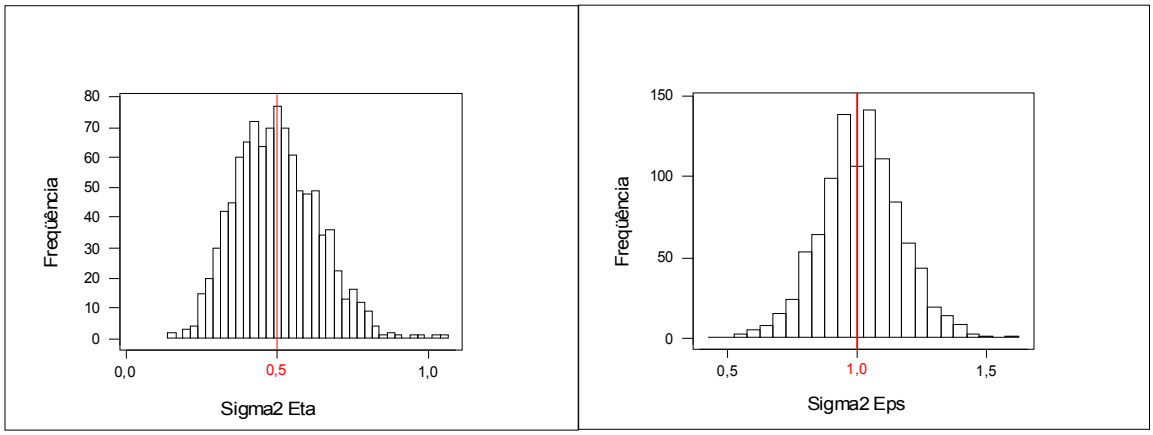

Figura 3.2: Histogramas das estimativas *bootstrap* do MNL de uma simulação MC

Visualizando a Figura 3.2, percebe-se que a maioria das estimativas *bootstrap* concentra-se em torno do verdadeiro valor do hiperparâmetro (indicado pelas linhas verticais nos gráficos). Na distribuição das estimativas do σ_{η}^{2} há indicação da presença de uma leve assimetria à direita.

Tabela 3.2: Intervalos de confiança *bootstrap* percentílico para os hiperparâmetros do MNL

Tamanho da série	Hiperparâmetro	Intervalo de Confiança *Bootstrap* Percentílico			
		Limite inferior	Limite superior	Amplitude	*Cobertura* (γ=*0,95*)
50	$\sigma_\eta^2 = 0{,}50$	0,0999	1,2683	1,1684	0,91
	$\sigma_\varepsilon^2 = 1{,}00$	0,3409	1,6301	1,2892	0,90
100	$\sigma_\eta^2 = 0{,}50$	0,1994	0,9991	0,7996	0,93
	$\sigma_\varepsilon^2 = 1{,}00$	0,5454	1,4544	0,9090	0,93
200	$\sigma_\eta^2 = 0{,}50$	0,2740	0,8045	0,5305	0,93
	$\sigma_\varepsilon^2 = 1{,}00$	0,6899	1,3199	0,6299	0,93

Na Tabela 3.2 são apresentados os intervalos de confiança para os hiperparâmetros do MNL, assim como suas amplitudes e suas coberturas. O nível de confiança adotado para a construção dos intervalos de confiança *bootstrap* é de 0,95, sendo que a cobertura dos mesmos se aproxima bastante desta confiança nominal. É possível observar também, que quando se aumenta o tamanho da série de *n=50* para *n=100*, há o aumento da cobertura dos intervalos, se aproximando ainda mais do nível de confiança adotado. Antes mesmo de se adotar o maior valor *n=200*, percebe-se a estabilização da cobertura dos intervalos. Com o aumento dos tamanhos das séries simuladas, também é observada a queda das amplitudes dos intervalos.

3.6.2. *Bootstrap* para o MTL

Através das rotinas computacionais apresentadas no Apêndice A, obteve-se os resultados apresentados na Tabela 3.3.

Tabela 3.3: EMV e estimativas *bootstrap* dos hiperparâmetros do MTL

Tamanho da série	Hiperparâmetro	EMV			Estimativas *Bootstrap*		
		Média	Vício (%)	EQM	Média*	Vício* (%)	EQM*
50	$\sigma_\eta^2 = 0{,}50$	0,6292	25,8	0,4625	0,6535	30,7	0,2918
	$\sigma_\xi^2 = 0{,}10$	0,0870	-13,0	0,0051	0,0950	-5,0	0,0045
	$\sigma_\varepsilon^2 = 1{,}00$	0,9712	-2,9	0,1699	0,9676	-3,2	0,1406
100	$\sigma_\eta^2 = 0{,}50$	0,5732	14,6	0,2599	0,6093	21,9	0,2270
	$\sigma_\xi^2 = 0{,}10$	0,0959	-4,1	0,0031	0,0990	-1,0	0,0028
	$\sigma_\varepsilon^2 = 1{,}00$	0,9646	-3,5	0,0935	0,9493	-5,1	0,0900
200	$\sigma_\eta^2 = 0{,}50$	0,5019	0,4	0,1205	0,5093	1,9	0,1096
	$\sigma_\xi^2 = 0{,}10$	0,0982	-1,8	0,0016	0,1008	0,8	0,0015
	$\sigma_\varepsilon^2 = 1{,}00$	1,0064	0,6	0,0442	1,0040	0,4	0,0433

Na Tabela 3.3, observa-se que as estimativas *bootstrap* obtidas estão próximas das EMV's, em especial, no caso em que se utiliza *n=200*. Para o hiperparâmetro σ_ξ^2, a estimativa *bootstrap* apresenta menor vício que a EMV para todos os tamanhos de séries assumidos. Para o hiperparâmetro σ_η^2, as EMV's superam as estimativas *bootstrap* em todos os tamanhos de séries simuladas, apresentando vícios inferiores. Com o aumento do tamanho da série simulada, observa-se uma maior aproximação das EMV's e estimativas *bootstrap* para o hiperparâmetro σ_ε^2. O comportamento das estimativas pode ser melhor visualizado na Figura 3.3.

Ao contrário do que foi observado no MNL, na Tabela 3.3 percebe-se que nos três tamanhos de série analisados, os EQM's obtidos utilizando-se a técnica de reamostragem *bootstrap* são menores que aqueles estimados por máxima

verossimilhança. À medida que se aumenta o tamanho da série, diminuem os EQM's, tanto nas EMV's quanto nas estimativas *bootstrap*.

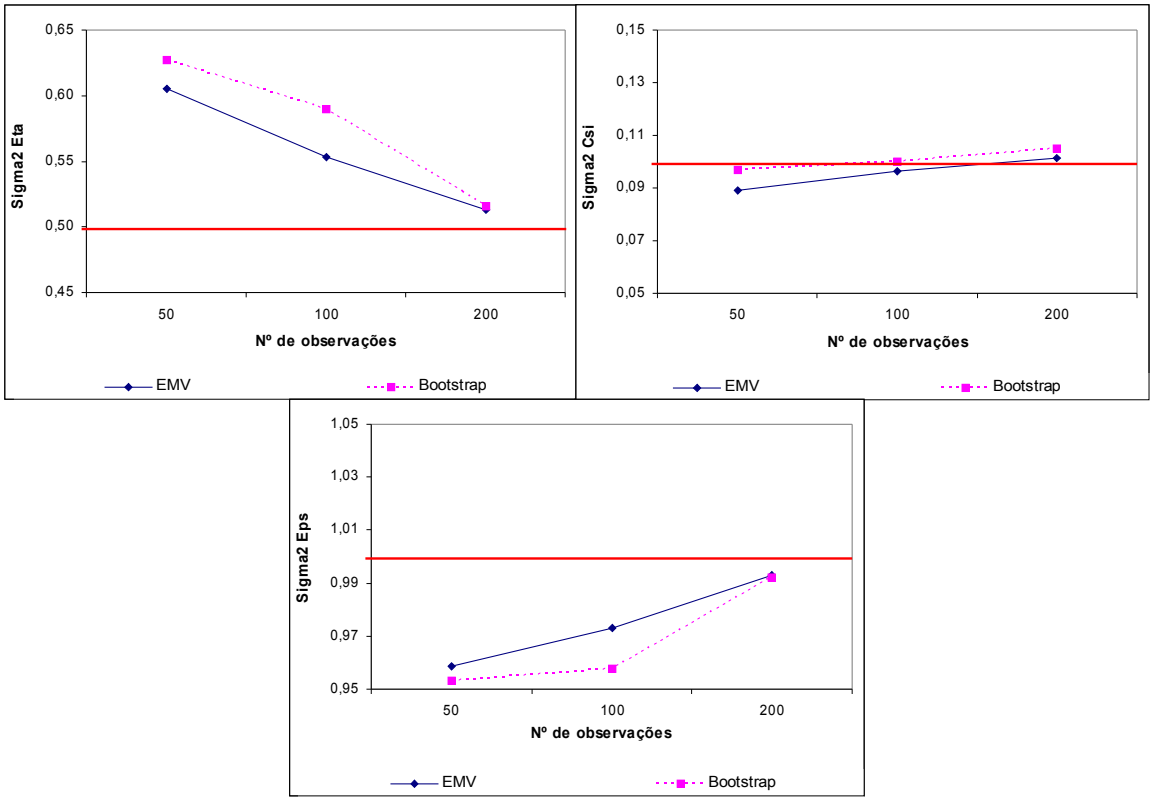

Figura 3.3: Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MTL

Tomando-se *1.000* estimativas *bootstrap* de uma das simulações, construíram-se histogramas a fim de analisar a forma da distribuição das estimativas de cada hiperparâmetro do MTL.

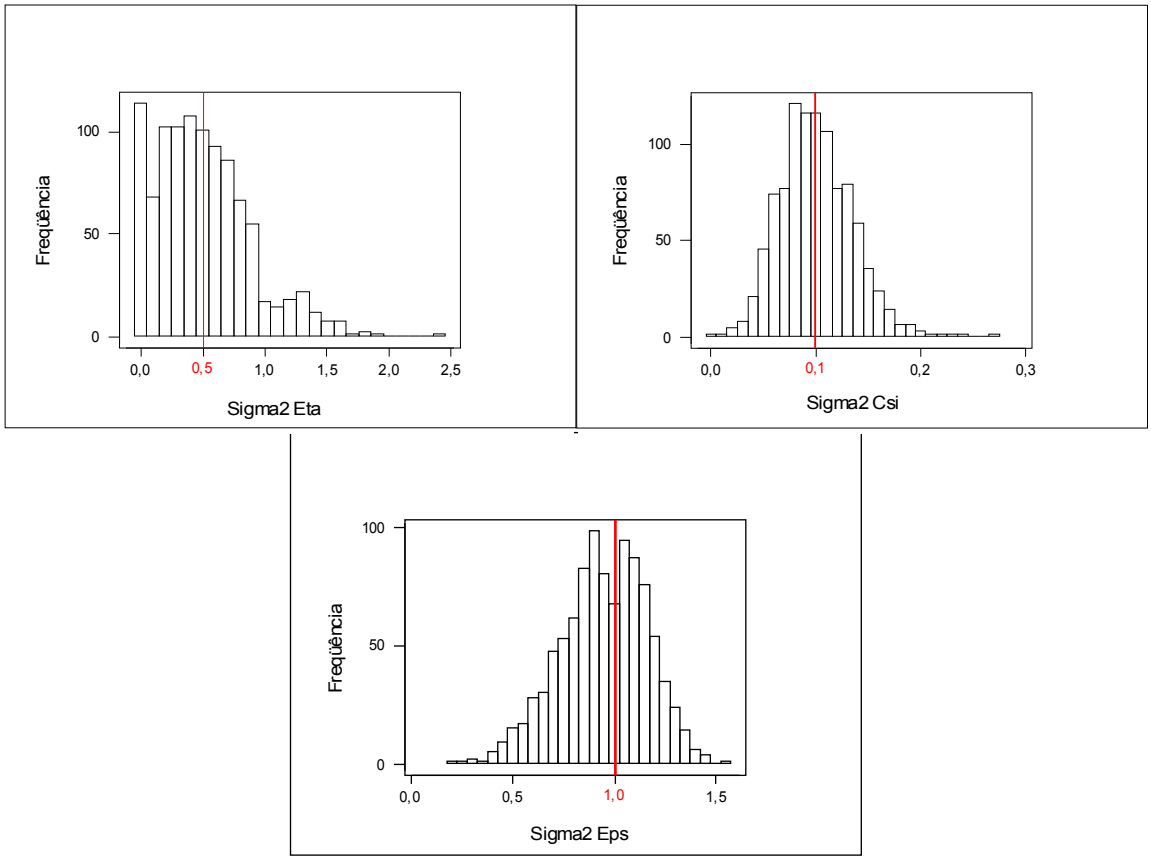

Figura 3.4: Histogramas das estimativas *bootstrap* do MTL de uma simulação MC

Visualizando a Figura 3.4, percebe-se que a maioria das estimativas *bootstrap* do hiperparâmetro σ_{η}^{2} concentra-se abaixo do valor predefinido (indicado pelo linha vertical), acusando a assimetria da distribuição. É interessante destacar que, dentre as estimativas *bootstrap* para o tamanho de série *n=200* (Tabela 3.3), o maior vício observado é para o hiperparâmetro σ_{η}^{2}, justificando a maior dispersão no histograma. As estimativas *bootstrap* do hiperparâmetro σ_{ξ}^{2} se distribuem com uma ligeira assimetria à direita. Já o histograma das estimativas *bootstrap* de σ_{ε}^{2} parece indicar a presença de uma distribuição bimodal.

Tabela 3.4: Intervalos de confiança *bootstrap* percentílico para os hiperparâmetros do MTL

Tamanho da série	Hiperparâmetro	Intervalo de Confiança *Bootstrap* Percentílico			
		Limite inferior	Limite superior	Amplitude	*Cobertura* (γ=*0,95*)
50	$\sigma_{\eta}^{2} = 0{,}50$	0,0007	2,1160	2,1153	0,99
	$\sigma_{\xi}^{2} = 0{,}10$	0,0015	0,2701	0,2686	0,88
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,1704	1,7647	1,5943	0,97
100	$\sigma_{\eta}^{2} = 0{,}50$	0,0265	1,6949	1,6684	0,98
	$\sigma_{\xi}^{2} = 0{,}10$	0,0161	0,2159	0,1998	0,88
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,3593	1,5320	1,1727	0,96
200	$\sigma_{\eta}^{2} = 0{,}50$	0,0460	1,2881	1,2421	0,97
	$\sigma_{\xi}^{2} = 0{,}10$	0,0382	0,1874	0,1492	0,94
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,5609	1,4029	0,8420	0,97

O nível de confiança adotado para a construção dos intervalos de confiança *bootstrap* é de 0,95, sendo que a cobertura dos mesmos se aproxima do valor nominal. Observa-se também, que as amplitudes dos intervalos de confiança diminuem à medida que se aumenta o tamanho da série simulada.

3.6.3. *Bootstrap* para o MEB

Através das rotinas computacionais apresentadas no Apêndice A, obtiveram-se os resultados apresentados na Tabela 3.5.

Tabela 3.5: EMV e estimativas *bootstrap* dos hiperparâmetros do MEB

Tamanho da série	Hiperparâmetro	EMV			Estimativas *Bootstrap*		
		Média	Vício (%)	EQM	Média*	Vício* (%)	EQM*
50	$\sigma_\eta^2 = 0{,}50$	0,5023	0,5	0,1807	0,4621	-7,6	0,1484
	$\sigma_\xi^2 = 0{,}01$	0,0107	7,2	0,0002	0,0154	54,2	0,0002
	$\sigma_\omega^2 = 0{,}10$	0,1089	8,9	0,0090	0,0963	-3,7	0,0059
	$\sigma_\varepsilon^2 = 1{,}00$	0,9587	-4,1	0,2546	1,1085	10,9	0,2795
100	$\sigma_\eta^2 = 0{,}50$	0,4989	-0,2	0,0984	0,4639	-7,2	0,1011
	$\sigma_\xi^2 = 0{,}01$	0,0109	8,9	0,0001	0,0137	37,2	0,0001
	$\sigma_\omega^2 = 0{,}10$	0,0998	-0,2	0,0036	0,0932	-6,8	0,0032
	$\sigma_\varepsilon^2 = 1{,}00$	1,0159	1,6	0,1284	1,1140	11,4	0,1666
200	$\sigma_\eta^2 = 0{,}50$	0,4924	-1,5	0,0459	0,4592	-8,2	0,0510
	$\sigma_\xi^2 = 0{,}01$	0,0106	6,1	0,0000	0,0123	23,4	0,0001
	$\sigma_\omega^2 = 0{,}10$	0,1000	0,0	0,0015	0,0966	-3,4	0,0014
	$\sigma_\varepsilon^2 = 1{,}00$	0,9921	-0,8	0,0553	1,0503	5,0	0,0721

Observando os vícios percentuais das estimativas na Tabela 3.5, em geral observa-se através das estimativas dos hiperparâmetros que as EMV's estão mais próximas dos valores reais assumidos do que as estimativas *bootstrap*. Como está apresentado na Figura 3.5, as melhores estimativas dos hiperparâmetros são obtidas quando se adota *n=200*, exceto nas estimativas do hiperparâmetro σ_η^2. Ainda assim, vale ressaltar que, adotando-se *n=200*, as EMV's são mais precisas que as estimativas *bootstrap* de todos os hiperparâmetros.

Diferentemente dos demais modelos analisados, a Tabela 3.5 mostra que, os EQM's obtidos utilizando-se a técnica de reamostragem *bootstrap* nem sempre são menores que aqueles estimados por máxima verossimilhança. Visualiza-se, também neste

modelo, a queda dos EQM's dos quatro hiperparâmetros, à medida que se aumenta o tamanho da série, tanto nas EMV's quanto nas estimativas *bootstrap*.

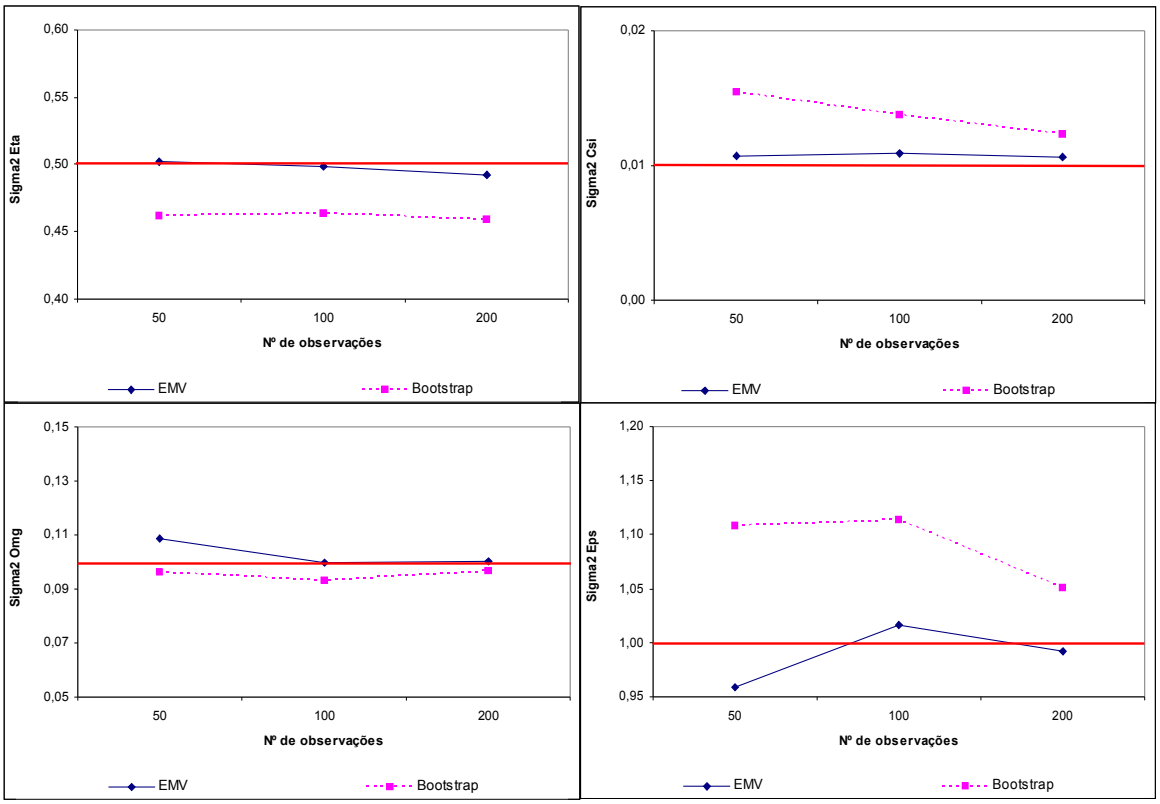

Figura 3.5: Gráficos das EMV e estimativas *bootstrap* dos hiperparâmetros do MEB

Tomando-se *1.000* estimativas *bootstrap* de uma das simulações, construíram-se histogramas a fim de analisar a forma da distribuição das estimativas de cada hiperparâmetro do MEB.

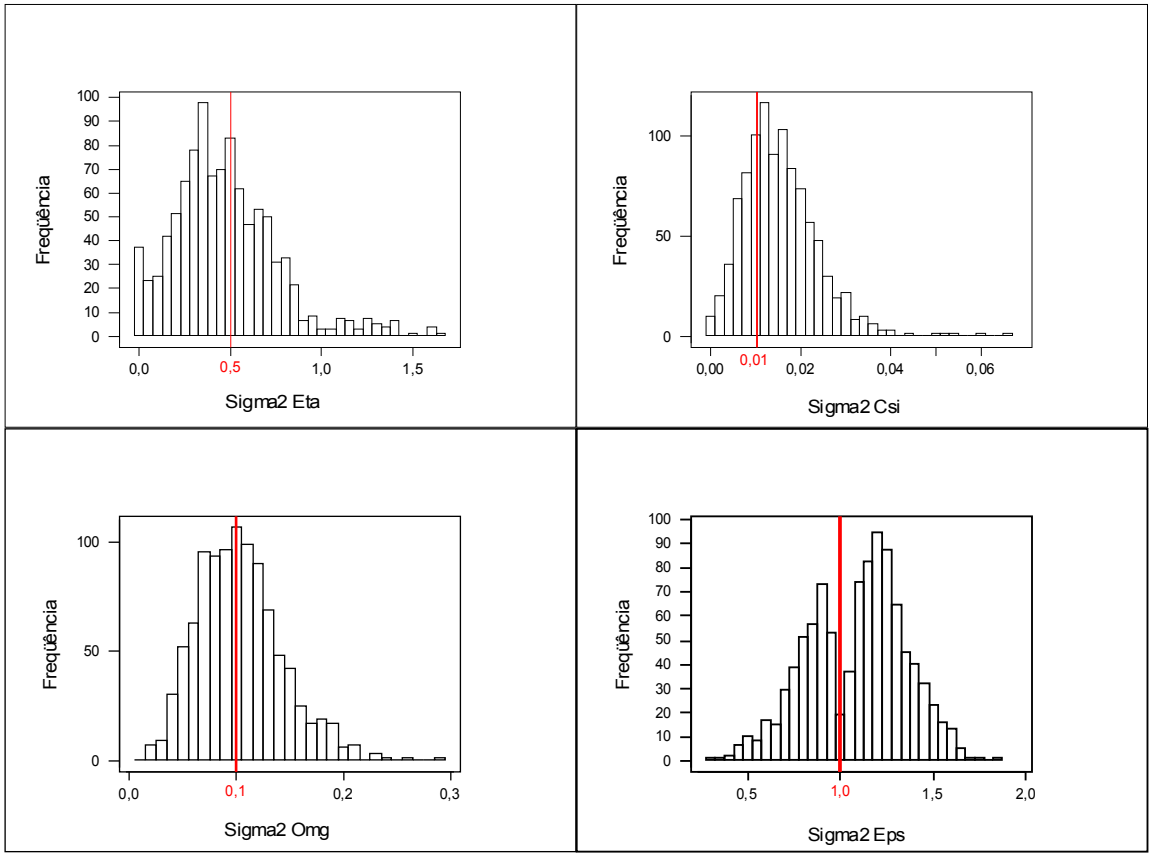

Figura 3.6: Histogramas das estimativas *bootstrap* do MEB de uma simulação MC

Através da Figura 3.6, percebe-se que a maioria das estimativas *bootstrap* do hiperparâmetro σ_{η}^{2} ficam dispersas em torno do valor real (indicado pelo linha vertical), acusando a assimetria da distribuição. Corroborando o fato de que a estimativa *bootstrap* do hiperparâmetro σ_{ξ}^{2} indica vício positivo, pode-se visualizar no seu histograma, que a maioria das suas estimativas está realmente acima do valor real. As estimativas *bootstrap* de σ_{ω}^{2} concentram-se em torno do valor real do hiperparâmetro, mas ainda assim há uma ligeira assimetria na distribuição. Já o histograma das estimativas *bootstrap* de σ_{ε}^{2} parece

indicar uma distribuição bimodal. A maioria das estimativas está acima do valor real do hiperparâmetro, confirmando o vício positivo obtido para σ_{ε}^{2}.

Tabela 3.6: Intervalos de confiança *bootstrap* percentílico para os hiperparâmetros do MEB

Tamanho da série	Hiperparâmetro	Intervalo de Confiança *Bootstrap* Percentílico			
		Limite inferior	Limite superior	Amplitude	*Cobertura* (γ=*0,95*)
50	$\sigma_{\eta}^{2} = 0{,}50$	0,0007	1,5767	1,5760	0,86
	$\sigma_{\xi}^{2} = 0{,}01$	0,0000	0,0586	0,0585	0,94
	$\sigma_{\omega}^{2} = 0{,}10$	0,0011	0,3014	0,3003	0,87
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,1520	2,1476	1,9957	0,95
100	$\sigma_{\eta}^{2} = 0{,}50$	0,0268	1,2460	1,2192	0,89
	$\sigma_{\xi}^{2} = 0{,}01$	0,0004	0,0396	0,0392	0,94
	$\sigma_{\omega}^{2} = 0{,}10$	0,0110	0,2203	0,2092	0,87
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,3606	1,8494	1,4888	0,94
200	$\sigma_{\eta}^{2} = 0{,}50$	0,0971	0,9648	0,8677	0,92
	$\sigma_{\xi}^{2} = 0{,}01$	0,0017	0,0284	0,0267	0,91
	$\sigma_{\omega}^{2} = 0{,}10$	0,0339	0,1800	0,1461	0,92
	$\sigma_{\varepsilon}^{2} = 1{,}00$	0,5394	1,5546	1,0152	0,94

O nível de confiança adotado para a construção dos intervalos de confiança *bootstrap* é de 0,95, sendo que a cobertura dos mesmos se aproxima do valor nominal. Observa-se também que as amplitudes dos intervalos de confiança diminuem à medida que se aumenta o tamanho da série simulada.

3.7. Conclusões e Observações Finais

Neste Capítulo foi apresentada uma pequena revisão sobre o método de reamostragem *bootstrap*. Com o conhecimento das definições apresentadas, tornou-se possível a aplicação de tal técnica, a fim de fazer inferência sobre os hiperparâmetros dos modelos estruturais, que é um dos objetivos desta dissertação.

Realizadas as implementações computacionais e as análises dos resultados na Seção 3.5, notou-se a precisão do processo de estimação dos hiperparâmetros, utilizando o método de máxima verossimilhança e também quando a técnica do *bootstrap* foi usada.

As conclusões apresentadas na Seção 2.7, a respeito da satisfação e precisão do processo de estimação implementado em Ox, foram confirmadas para as EMV's, assim como para as estimativas *bootstrap*. Para os três modelos ajustados, a implementação em Ox forneceu baixos vícios nas estimativas dos valores esperados dos hiperparâmetros e pequenos erros quadráticos médios (EQM).

Mediante a construção dos histogramas das estimativas *bootstrap* dos hiperparâmetros, notou-se a presença de assimetria nas distribuições das mesmas. Houve também uma sutil indicação de que, quanto mais complexo o modelo, mais dispersas ficaram as estimativas dos hiperparâmetros.

Quanto aos intervalos de confiança *bootstrap*, percebeu-se que a cobertura dos mesmos ficou próxima do nível de confiança nominal. Notou-se também que, quanto maior o tamanho das séries simuladas, menores eram as amplitudes dos intervalos calculados.

Em geral, também foi possível observar que, na maioria das simulações em que se adotou $n=200$, os vícios e os EQM's das estimativas dos hiperparâmetros (tanto nas EMV's quanto nas estimativas *bootstrap*) foram menores que nos demais tamanhos de séries estudados. Desta forma, pode-se concluir que os estimadores analisados são assintoticamente não-viesados.

Diante dos resultados apresentados nesta seção, foi possível verificar a semelhança das EMV's e as estimativas *bootstrap* na grande maioria dos casos, indicando que o processo de estimação utilizando a técnica de reamostragem *bootstrap* é confiável.

4. Aplicação a Série Real

Neste Capítulo, a metodologia apresentada nas seções anteriores será aplicada a uma série temporal real. Novos programas computacionais foram implementados na linguagem Ox (vide Apêndice C), fazendo a modelagem e a aplicação da metodologia *bootstrap* em séries temporais reais.

Uma série histórica contendo as variações mensais do Índice de Preços ao Consumidor Amplo de Belo Horizonte, calculada pela Fundação IPEAD (www.ipead.face.ufmg.br/ipc), no período entre janeiro de 1997 e outubro de 2005 (vide Apêndice D), foi analisada.

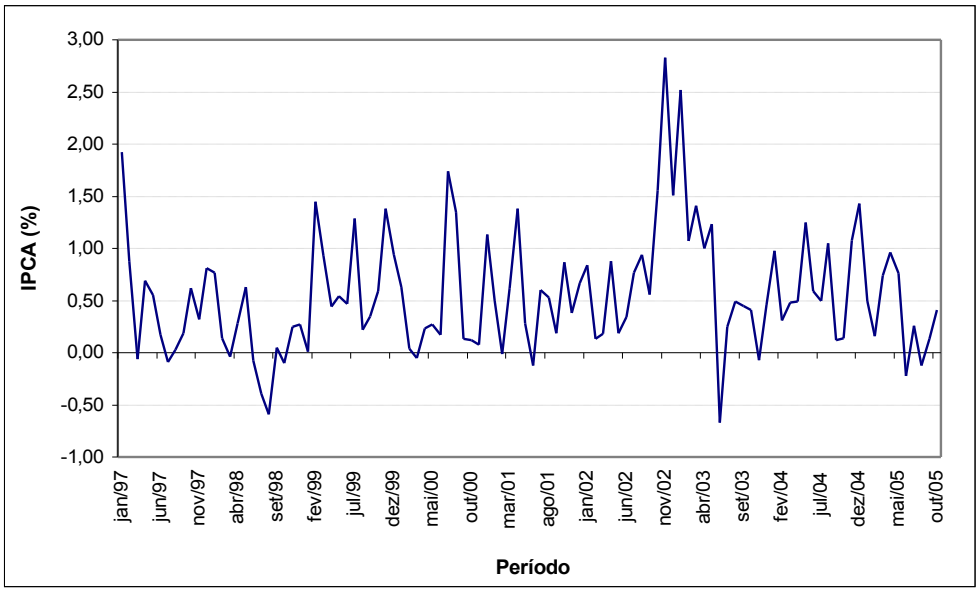

Figura 4.1: Série temporal do IPCA (%) de Belo Horizonte (jan/97 a out/05)

Na Figura 4.1, percebe-se que os dados temporais no período analisado não apresentaram nenhum tipo de sazonalidade e nenhuma tendência explícita de crescimento/decrescimento. As observações desta série temporal caracterizam um MNL, pois flutuam aleatoriamente em torno de um mesmo nível.

Tabela 4.1: Análise descritiva da série do IPCA (%) de Belo Horizonte

Variável	N	Média	Desvio Padrão	Mínimo	Máximo	1º Quartil	Mediana	3º Quartil
IPCA (%)	106	0,56	0,58	-0,67	2,83	0,16	0,48	0,88

A série temporal do IPCA (%) contém 106 observações cujo valor médio é 0,56 e o desvio padrão é 0,58. O valor mínimo observado é -0,67 e o valor máximo é 2,83. A metade dos dados está contida entre os valores 0,16 e 0,88. A mediana igual a 0,48 indica que 50% dos dados estão acima/abaixo deste valor, como também é possível observar através da Figura 4.2.

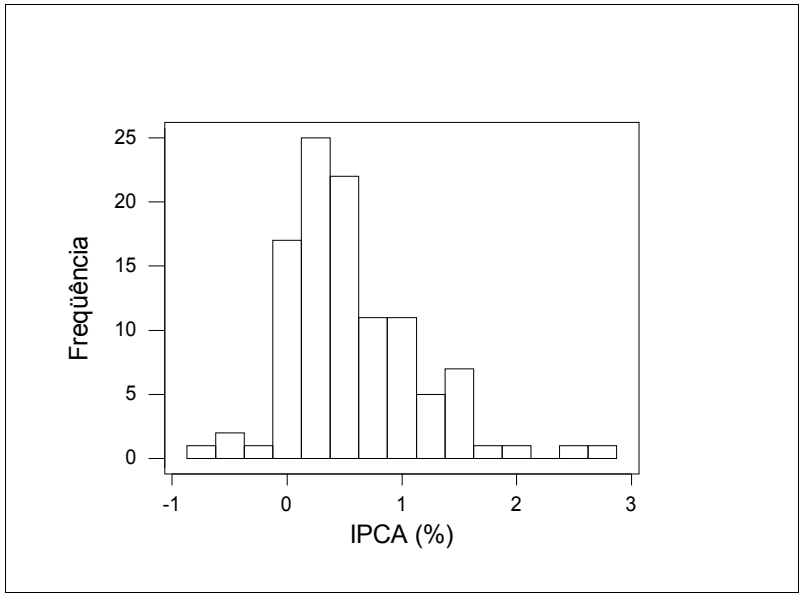

Figura 4.2: Histograma da série do IPCA (%) de Belo Horizonte

Mesmo sabendo que o modelo ideal para este conjunto de dados é o MNL, a princípio tentou-se ajustar o MEB para observar o comportamento das estimativas fornecidas pela implementação em Ox e para verificar a possibilidade de utilizar o *bootstrap* para se fazer inferência sobre o modelo correto a ser adotado neste caso. Os resultados obtidos estão apresentados na Tabela 4.2.

Tabela 4.2: EMV e estimativas *bootstrap* dos hiperparâmetros do MEB para a série do IPCA (%)

Hiperparâmetro	EMV	Estimativa *bootstrap*	Intervalo de Confiança *Bootstrap* Percentílico (γ=0,95)	
			Limite Inferior	Limite Superior
σ_{η}^{2}	0,0444	0,0686	0,0000	0,4070
σ_{ξ}^{2}	0,0000	0,0001	0,0000	0,0005
σ_{ω}^{2}	0,0000	0,0016	0,0000	0,0102
σ_{ε}^{2}	0,1720	0,1558	0,0000	0,2569

Através dos resultados apresentados na Tabela 4.2, percebe-se que as EMV's dos hiperparâmetros σ_{ω}^{2} e σ_{ξ}^{2} são zero, as estimativas *bootstrap* também estão próximas de zero e os intervalos de confiança obtidos parecem indicar que não há presença de sazonalidade e nem tendência na série temporal do IPCA (%). Inicialmente, será retirado somente o hiperparâmetro σ_{ω}^{2} para analisar o ajuste de um MTL para esta série.

Tabela 4.3: EMV e estimativas *bootstrap* dos hiperparâmetros do MTL para a série do IPCA (%)

Hiperparâmetro	EMV	Estimativa *bootstrap*	Intervalo de Confiança *Bootstrap* Percentílico (γ=0,95)	
			Limite Inferior	Limite Superior
σ_{η}^{2}	0,0502	0,0447	0,0000	0,1044
σ_{ξ}^{2}	0,0000	0,0001	0,0000	0,0007
σ_{ε}^{2}	0,1984	0,1992	0,1213	0,2996

Na Tabela 4.3, percebe-se que a EMV do hiperparâmetro σ_{ξ}^{2} também é zero e a estimativa *bootstrap* é muito próxima de zero, com o intervalo de confiança *bootstrap* percentílico obtido possuindo limites inferiores e superiores indicando a não existência de tendência na série analisada. Percebe-se também que as estimativas dos demais hiperparâmetros, através das EMV's e das estimativas *bootstrap* se aproximam depois da retirada do σ_{ω}^{2}. Retirando o hiperparâmetro σ_{ξ}^{2}, finalmente o modelo ideal para tais dados (MNL) será ajustado.

Tabela 4.4: EMV e estimativas *bootstrap* dos hiperparâmetros do MNL para a série do IPCA (%)

Hiperparâmetro	EMV	Estimativa *bootstrap*	Intervalo de Confiança *Bootstrap* Percentílico *(γ=0,95)*	
			Limite Inferior	Limite Superior
σ_{η}^{2}	0,0423	0,0427	0,0112	0,0875
σ_{ε}^{2}	0,2063	0,2036	0,1248	0,3014

Na Tabela 4.4, percebe-se que as EMV's e as estimativas *bootstrap* não ficam muito próximas de zero como nos outros ajustes, sendo que os intervalos de confiança obtidos não contêm o valor nulo. Este fato indica a presença dos hiperparâmetros σ_{η}^{2} e σ_{ε}^{2}, sugerindo que o MNL é o melhor modelo ajustado aos dados. Percebe-se também que, após a retirada dos hiperparâmetros nulos σ_{ω}^{2} e σ_{ξ}^{2}, as EMV's e as estimativas *bootstrap* do σ_{η}^{2} e σ_{ε}^{2} tornam-se quase idênticas.

5. Conclusões Finais

Ao longo da realização desta dissertação, tornou-se possível conhecer melhor os modelos estruturais e, por meio deles, alcançou-se, com sucesso, todos os objetivos almejados.

No estudo preliminar da Seção 2.6, a realização de estudos Monte Carlo foi de suma importância, ao possibilitar a identificação de fatores influentes nas implementações. Desde então, os resultados simulados já se mostravam precisos com a utilização de implementações na linguagem Ox. Em geral, as estimativas obtidas dos valores esperados dos hiperparâmetros ficaram bem próximas dos seus valores reais e pequenos EQM's foram observados. Foi interessante verificar que, adotando-se o maior tamanho da série, as estimativas dos hiperparâmetros se aproximavam mais dos valores predefinidos, podendo-se concluir que tais estimadores são consistentes.

No Capítulo 3, a técnica de reamostragem *bootstrap* foi abordada, apresentando-se a sua aplicabilidade a fim de fazer inferência sobre os hiperparâmetros dos modelos estruturais. Mediante os resultados apresentados na Seção 3.5, o fato de que as implementações em Ox forneciam estimativas satisfatórias dos hiperparâmetros foi corroborado. Notou-se a eficiência do processo de estimação, utilizando-se tanto o método de máxima verossimilhança quanto a técnica de reamostragem *bootstrap*.

Por meio dos histogramas das estimativas *bootstrap* dos hiperparâmetros, a presença de assimetria nas distribuições foi perceptível, assim como uma maior dispersão das estimativas, na medida em que o modelo ficava mais complexo.

A respeito dos intervalos de confiança *bootstrap* dos hiperparâmetros, percebeu-se que, com o aumento do tamanho das séries simuladas, as amplitudes intervalares decresceram. As coberturas dos intervalos de confiança se mantiveram próximas do nível de confiança assumido.

Concluiu-se, novamente, que os estimadores analisados são assintoticamente não-viesados, pois, na maioria das simulações em que se adotou $n=200$, os vícios e os EQM's das estimativas dos hiperparâmetros (tanto nas EMV's quanto nas estimativas *bootstrap*) foram os menores observados.

No Capítulo 4 foi apresentada a aplicação da metodologia estudada em uma série temporal real. Tomou-se uma série cujo comportamento caracteriza-se como sendo um passeio aleatório. Ajustou-se o MEB e o MTL com a intenção de verificar se a implementação captaria a não existência dos componentes de tendência e de sazonalidade. Em ambos os ajustes, tal objetivo foi alcançado, verificando-se a nulidade das estimativas dos hiperparâmetros de tais componentes. O modelo apropriado para tal série (MNL) foi ajustado e observou-se a grande proximidade entre as EMV's e as estimativas *bootstrap*.

Diante da apresentação dos resultados obtidos, concluiu-se que a implementação da estimação dos hiperparâmetros dos modelos estruturais utilizando a linguagem Ox mostrou-se satisfatória, tanto em dados simulados, quanto nos dados da série temporal real. Por fim, tendo em vista a grande semelhança das EMV's e as estimativas *bootstrap*, foi possível confirmar a grande precisão do processo de estimação utilizando a técnica de reamostragem *bootstrap*.

Tópicos para pesquisas futuras na área incluem a ampliação dos estudos de simulação para modelos incompletos, a análise de outras séries reais, bem como a implementação de testes de hipóteses para testar a significância dos hiperparâmetros.

APÊNDICE A

A.1 BIBLIOTECA "STSBOOTPACK.OX" EM OX PARA IMPLEMENTAR OS ESTIMADORES DE MÁXIMA VEROSSIMILHANÇA (EMV), OS ESTIMADORES *BOOTSTRAP* E INTERVALOS DE CONFIANÇA *BOOTSTRAP* PARA OS HIPERPARÂMETROS DOS MODELOS ESTRUTURAIS

```
////////////////////////////////////////////////////////////////
// including libraries
////////////////////////////////////////////////////////////////
#include <oxstd.h>
#include <packages\\SsfPack\\Ssfpack.h>
#import  <maximize>
```

```
////////////////////////////////////////////////////////////////
// declaring globals
////////////////////////////////////////////////////////////////
decl g_STSiBoot=1000;
decl g_STSiBFGS=50;
decl g_STSiBurnIn=100;
decl g_STSiMessages=FALSE;
static decl g_STSvYt;
static decl g_STSmPhi, g_STSmOmega, g_STSmSigma, g_STSmKF;
```

```
////////////////////////////////////////////////////////////////
// declaring headers
////////////////////////////////////////////////////////////////
STSLoglike(const vP, vFunc, const vScore, const mHess);
STSAdjust(const vYt, const mSTSModel);
STSBoot(const vYt, const mSTSModel);
STSBootCI(const vYt, const mSTSModel);
STSQuantile(const vInp, const fQuant);
STSSimulateLLM(const dSigma2Eta, const dSigma2Eps,
        const iYtSize);
STSSimulateLTL(const dSigma2Eta, const dSigma2Csi,
        const dSigma2Eps, const iYtSize);
STSSimulateSBM(const dSigma2Eta, const dSigma2Csi,
        const dSigma2Eps, const dSigma2Omg,
        const iSazo, const iYtSize);
```

```
////////////////////////////////////////////////////////////////
// implementing
////////////////////////////////////////////////////////////////
// defining Log-likelihood function
////////////////////////////////////////////////////////////////
STSLoglike(const vP, vFunc, const vScore, const mHess) {
//      println("STSLoglike:STSLoglike()");
        decl ii, dVar;
        ii=rows(g_STSmOmega);
        if (ii==2) {
                g_STSmOmega[0][0]=exp(2.0*vP[0]);
                g_STSmOmega[1][1]=exp(2.0*vP[1]);
        } else if (ii==3) {
                g_STSmOmega[0][0]=exp(2.0*vP[0]);
                g_STSmOmega[1][1]=exp(2.0*vP[1]);
                g_STSmOmega[2][2]=exp(2.0*vP[2]);
        } else {
                g_STSmOmega[0][0]=exp(2.0*vP[0]);
                g_STSmOmega[1][1]=exp(2.0*vP[1]);
                g_STSmOmega[2][2]=exp(2.0*vP[2]);
                g_STSmOmega[ii-1][ii-1]=exp(2.0*vP[3]);
        }
        ii=SsfLik(vFunc, &dVar,
                g_STSvYt, g_STSmPhi, g_STSmOmega, g_STSmSigma);
//      println("STSLoglike:vP =", vP);
//      println("STSLoglike:g_STSmOmega=", g_STSmOmega);
//      println("STSLoglike:vFunc =", vFunc);
//      println("STSLoglike:dVar=", dVar);
        return(ii);
}
```

```
//////////////////////////////////////////////////////////////
// adjusting the model
//////////////////////////////////////////////////////////////
STSAdjust(const vYt, const mSTSModel) {
//      println("STSAdjust:STSAdjust()");
//      var declaration
        decl vP, vFunc, dVar;
        decl ii;
//      get time series
        g_STSvYt=vYt;
//      initialize independent variables
        GetSsfStsm(mSTSModel, &g_STSmPhi, &g_STSmOmega,
                &g_STSmSigma);
        ii=rows(g_STSmOmega);
        if (ii==2) {
                vP=zeros(2,1);
                vP[0]=0.5*log(g_STSmOmega[0][0]);
                vP[1]=0.5*log(g_STSmOmega[1][1]);
        } else if (ii==3) {
                vP=zeros(3,1);
                vP[0]=0.5*log(g_STSmOmega[0][0]);
                vP[1]=0.5*log(g_STSmOmega[1][1]);
                vP[2]=0.5*log(g_STSmOmega[2][2]);
        } else {
                vP=zeros(4,1);
                vP[0]=0.5*log(g_STSmOmega[0][0]);
                vP[1]=0.5*log(g_STSmOmega[1][1]);
                vP[2]=0.5*log(g_STSmOmega[2][2]);
                vP[3]=0.5*log(g_STSmOmega[ii-1][ii-1]);
        }
//      log-likelihood maximization
        MaxControl(g_STSiBFGS,-1);
        ii=MaxBFGS(STSLoglike, &vP, &vFunc, 0, TRUE);
        if ((ii!=FALSE)&&(g_STSiMessages==TRUE)) {
                println("STSAdjust:ii=", ii, " ", MaxConvergenceMsg(ii));
        }
//      println("STSAdjust:g_STSmPhi=", g_STSmPhi);
//      println("STSAdjust:g_STSmOmega=", g_STSmOmega);
//      println("STSAdjust:g_STSmSigma=", g_STSmSigma);
//      println("STSAdjust:vP=", vP);
//      println("STSAdjust:", MaxConvergenceMsg(ii),
//                      " using numerical derivatives\n");
//      println("STSAdjust:g_STSmOmega=", g_STSmOmega);
//      println("STSAdjust:vP=", vP);
//      println("STSAdjust:vFunc =", vFunc);
        return g_STSmOmega;
}
```

```
//////////////////////////////////////////////////////////////
// bootstrapping time series
//////////////////////////////////////////////////////////////
STSBoot(const vYt, const mSTSModel) {
//      println("STSBoot:STSBoot()");
//      var declaration
        decl mKF, mPhi, mOmega;
        decl vVt, vKt, vFt, vEt, vRandU;
        decl vAtMu, vAtBeta, vAtGamma;
        decl iYtSize, il, iSazo, i, j, vYtBoot;
        decl iBurnIn=g_STSiBurnIn;
//      model adjustment
        g_STSmOmega=STSAdjust(vYt, mSTSModel);
//      Kalman filter estimation
        g_STSmKF=KalmanFil(vYt, g_STSmPhi, g_STSmOmega);
//      bootstraping
        il=rows(g_STSmOmega);
        if (il==2) {
//              get Vt, Kt, Ft, and vEt
                iYtSize=columns(vYt);
                vVt=g_STSmKF[0][:];
                vKt=g_STSmKF[1][:];
                vFt=ones(1,iYtSize)./g_STSmKF[2][:];
//              vEt = (vVt-meanr(vVt))./(sqrt(vFt));
                vEt = (vVt)./(sqrt(vFt));
//              bootstraping redidues
                vRandU=idiv((iYtSize-1)*ranu(iYtSize,1),1)+1;
//              creating bootstrap time series
                vAtMu=zeros(1, iYtSize+1);
                vYtBoot=zeros(1, iYtSize);
                vAtMu[0]=0;
```

```
                        for (i=0; i<iYtSize; i++) {
                                vYtBoot[i]=vAtMu[i]+
                                        sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtMu[i+1]=vAtMu[i]+
                                        vKt[i]*sqrt(vFt[i])*vEt[vRandU[i]];
                        }
//                      print("STSBoot:g_STSmKF=", g_STSmKF);
//                      print("STSBoot:vVt=", vVt);
//                      print("STSBoot:vKt=", vKt);
//                      print("STSBoot:vFt=", vFt);
//                      print("STSBoot:vEt=", vEt);
//                      print("STSBoot:vRandU=", vRandU);
//                      print("STSBoot:vAtMu=", vAtMu);
//                      print("STSBoot:vYtBoot=", vYtBoot);
//                      exit (0);
                } else if (il==3) {
//                      get Vt, Kt, Ft, and vEt
                        iYtSize=columns(vYt);
                        vVt=g_STSmKF[0][:];
                        vKt=g_STSmKF[1:2][:];
                        vFt=ones(1,iYtSize)./g_STSmKF[3][:];
//                      vEt = (vVt-meanr(vVt))./(sqrt(vFt));
                        vEt = (vVt)./(sqrt(vFt));
//                      bootstraping redidues
                        vRandU=idiv((iYtSize-2)*ranu(iYtSize,1),1)+2;
//                      creating bootstrap time series
                        vAtMu=zeros(1, iYtSize+1);
                        vAtBeta=zeros(1, iYtSize+1);
                        vYtBoot=zeros(1, iYtSize);
                        vAtMu[0]=0;
                        vAtBeta[0]=0;
                        for (i=0; i<iYtSize; i++) {
                                vYtBoot[i]=vAtMu[i]+
                                        sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtMu[i+1]=vAtMu[i]+vAtBeta[i]+
                                        vKt[0][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtBeta[i+1]=vAtBeta[i]+
                                        vKt[1][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                        }
//                      print("STSBoot:g_STSmKF=", g_STSmKF);
```

```
//                      print("STSBoot:vVt=", vVt);
//                      print("STSBoot:vVt=");
//                      for (i=0; i<iYtSize; i++) {
//                              println(vVt[i]);
//                      }
//                      print("STSBoot:vKt=", vKt);
//                      print("STSBoot:vFt=", vFt);
//                      print("STSBoot:vEt=", vEt);
//                      print("STSBoot:vEt=");
//                      for (i=0; i<iYtSize; i++) {
//                              println(vEt[i]);
//                      }
//                      print("STSBoot:vRandU=", vRandU);
//                      print("STSBoot:vAtBeta=", vAtBeta);
//                      print("STSBoot:vAtMu=", vAtMu);
//                      print("STSBoot:vYtBoot=", vYtBoot);
//                      exit (0);
                } else if (il>3) {
//                      get Vt, Kt, Ft, and vEt
                        iYtSize=columns(vYt);
                        iSazo=il-3;
                        vVt=g_STSmKF[0][:];
                        vKt=g_STSmKF[1:il-1][:];
                        vFt=ones(1,iYtSize)./g_STSmKF[il][:];
//                      vEt = (vVt-meanr(vVt))./(sqrt(vFt));
                        vEt = (vVt)./(sqrt(vFt));
//                      bootstraping redidues
                        vRandU=idiv((iYtSize-il)*ranu(iYtSize,1),1)+il;
//                      creating bootstrap time series
                        vAtMu=zeros(1, iYtSize+1);
                        vAtBeta=zeros(1, iYtSize+1);
                        vAtGamma=zeros(il-3, iYtSize+1);
                        vYtBoot=zeros(1, iYtSize);
                        vAtMu[0]=0;
                        vAtBeta[0]=0;
                        for (j=0; j<iSazo; j++) {
                                vAtGamma[j][0]=0;
                        }
                        for (i=0; i<iYtSize; i++) {
                                vYtBoot[i]=vAtMu[i]+vAtGamma[0][i]+
```

```
                                        sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtMu[i+1]=vAtMu[i]+vAtBeta[i]+
                                        vKt[0][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtBeta[i+1]=vAtBeta[i]+
                                        vKt[1][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                                vAtGamma[0][i+1]=
                                        vKt[2][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                                for (j=0; j<iSazo; j++) {
                                        vAtGamma[0][i+1]=vAtGamma[0][i+1]-vAtGamma[j][i];
                                }
                                for (j=1; j<iSazo; j++) {
                                        vAtGamma[j][i+1]=vAtGamma[j-1][i]+
                                                vKt[j+2][i]*sqrt(vFt[i])*vEt[vRandU[i]];
                                }
                        }
//                      print("STSBoot:g_STSmKF=", g_STSmKF);
//                      print("STSBoot:vFt=", vFt);
//                      print("STSBoot:vVt=", vVt);
//                      println("STSBoot:vVt=");
//                      for (i=0; i<iYtSize; i++) {
//                              println(vVt[i]);
//                      }
//                      print("STSBoot:vKt=", vKt);
//                      print("STSBoot:vFt=", vFt);
//                      print("STSBoot:vEt=", vEt);
//                      println("STSBoot:vEt=");
//                      for (i=0; i<iYtSize; i++) {
//                              println(vEt[i]);
//                      }
//                      println("STSBoot:vRandU=", vRandU);
//                      print("STSBoot:vAtMu=", vAtMu);
//                      print("STSBoot:vAtBeta=", vAtBeta);
//                      print("STSBoot:vAtGamma=", vAtGamma);
//                      print("STSBoot:vYtBoot=", vYtBoot);
//                      exit (0);
        }
//      results
//      print("STSBoot:vYtBoot=", vYtBoot);
        return (vYtBoot);
}
```

```
//////////////////////////////////////////////////////////////////
// computing confidence intervals
//////////////////////////////////////////////////////////////////
STSBootCI(const vYt, const mSTSModel) {
        println("STSBootCI:STSBootCI()");
        decl i, j, il, vYtBoot, STSmOmega;
        decl fSigma2Eta=zeros(1,g_STSiBoot);
        decl fSigma2Csi=zeros(1,g_STSiBoot);
        decl fSigma2Omg=zeros(1,g_STSiBoot);
        decl fSigma2Eps=zeros(1,g_STSiBoot);
        decl mResult;
//      model adjustment
        g_STSmOmega=STSAdjust(vYt, mSTSModel);
//      bootstraping
        il=rows(g_STSmOmega);
        if (il==2) {
                for (i=0; i<g_STSiBoot; i++) {
                        vYtBoot=STSBoot(vYt, mSTSModel);
                        STSmOmega=STSAdjust(vYtBoot, mSTSModel);
                        fSigma2Eta[i]=STSmOmega[0][0];
                        fSigma2Eps[i]=STSmOmega[1][1];
//                      println("STSBootCI:STSmOmega=", STSmOmega);
//                      println("STSBootCI:fSigma2Eta=", fSigma2Eta[i]);
//                      println("STSBootCI:fSigma2Eps=", fSigma2Eps[i]);
//                      exit(0);
                        print(".");
                }
                println("");
                mResult=zeros(2,g_STSiBoot);
                mResult[0][:]=fSigma2Eta;
                mResult[1][:]=fSigma2Eps;
        } else if (il==3) {
                for (i=0; i<g_STSiBoot; i++) {
                        vYtBoot=STSBoot(vYt, mSTSModel);
                        STSmOmega=STSAdjust(vYtBoot, mSTSModel);
                        fSigma2Eta[i]=STSmOmega[0][0];
                        fSigma2Csi[i]=STSmOmega[1][1];
                        fSigma2Eps[i]=STSmOmega[2][2];
//                      println("STSBootCI:STSmOmega=", STSmOmega);
```

```
//                              println("STSBootCI:fSigma2Eta=", fSigma2Eta[i]);
//                              println("STSBootCI:fSigma2Csi=", fSigma2Csi[i]);
//                              println("STSBootCI:fSigma2Eps=", fSigma2Eps[i]);
//                              exit(0);
                                print(".");
                        }
                        println("");
                        mResult=zeros(3,g_STSiBoot);
                        mResult[0][:]=fSigma2Eta;
                        mResult[1][:]=fSigma2Csi;
                        mResult[2][:]=fSigma2Eps;
        } else if (il>3) {
                        for (i=0; i<g_STSiBoot; i++) {
                                vYtBoot=STSBoot(vYt, mSTSModel);
                                STSmOmega=STSAdjust(vYtBoot, mSTSModel);
                                fSigma2Eta[i]=STSmOmega[0][0];
                                fSigma2Csi[i]=STSmOmega[1][1];
                                fSigma2Omg[i]=STSmOmega[2][2];
                                fSigma2Eps[i]=STSmOmega[il-1][il-1];
//                              println("STSBootCI:STSmOmega=", STSmOmega);
//                              println("STSBootCI:fSigma2Eta=", fSigma2Eta[i]);
//                              println("STSBootCI:fSigma2Csi=", fSigma2Csi[i]);
//                              println("STSBootCI:fSigma2Omg=", fSigma2Omg[i]);
//                              println("STSBootCI:fSigma2Eps=", fSigma2Eps[i]);
//                              exit(0);
                                print(".");
                        }
                        println("");
                        mResult=zeros(4,g_STSiBoot);
                        mResult[0][:]=fSigma2Eta;
                        mResult[1][:]=fSigma2Csi;
                        mResult[2][:]=fSigma2Omg;
                        mResult[3][:]=fSigma2Eps;
        }
//      println("STSBootCI:mResult=", mResult);
        return (mResult);
}
```

```
////////////////////////////////////////////////////////////////
```

```
// computing quantiles
////////////////////////////////////////////////////////////////
STSQuantile(const vInp, const fQuant) {
//      println("STSQuantile:STSQuantile()");
        decl iSize, lowPos, highPos, position;
        decl vInpSort, i, j, jMin, min, aux, lowData, highData, fPerc;
        // sort data
        iSize=columns(vInp);
        vInpSort=vInp;
        for (i=0; i<iSize; i++) {
                min=vInpSort[i];
                jMin=i;
                for (j=i+1; j<iSize; j++) {
                        if (vInpSort[j]<min) {
                                min=vInpSort[j];
                                jMin=j;
                        }
                }
                aux=vInpSort[i];
                vInpSort[i]=vInpSort[jMin];
                vInpSort[jMin]=aux;
        }
        // find positions
        lowPos = trunc((iSize-1)*fQuant)+1;
        highPos = trunc((iSize-1)*fQuant+0.999999)+1;
        position = (iSize-1)*fQuant+1;
        // find data
        lowData=vInpSort[lowPos-1];
        highData=vInpSort[highPos-1];
        // percentile
        fPerc = lowData + (highData-lowData)*(position-lowPos);
//      println("STSQuantile():iSize=", iSize);
//      println("STSQuantile():vInpSort=", vInpSort);
//      println("STSQuantile():lowPos=", lowPos);
//      println("STSQuantile():highPos=", highPos);
//      println("STSQuantile():position=", position);
//      println("STSQuantile():lowData=", lowData);
//      println("STSQuantile():highData=", highData);
//      println("STSQuantile():fPerc=", fPerc);
        return(fPerc);
```

```
}
```

```
////////////////////////////////////////////////////////////////
// simulate time series
////////////////////////////////////////////////////////////////
// Local Level Model
////////////////////////////////////////////////////////////////
STSSimulateLLM(const dSigma2Eta, const dSigma2Eps,
                                                const iYtSize) {
//          println("STSSimulateLLM:STSSimulateLLM()");
            decl iBurnIn=g_STSiBurnIn;
            decl vEta=sqrt(dSigma2Eta)*rann(1,iYtSize+iBurnIn);
            decl vEps=sqrt(dSigma2Eps)*rann(1,iYtSize+iBurnIn);
            decl vMu=zeros(1,iYtSize+iBurnIn);
            decl vYtAux=zeros(1,iYtSize+iBurnIn);
            decl vYt=zeros(1,iYtSize);
            decl i;
            vMu[0]=vEta[0];
            vYtAux[0]=vMu[0]+vEps[0];
            for (i=1; i<iYtSize+iBurnIn; i++) {
               vMu[i]=vMu[i-1]+vEta[i];
                        vYtAux[i]=vMu[i]+vEps[i];
            }
            for (i=0; i<iYtSize; i++) {
                        vYt[i]=vYtAux[i+iBurnIn];
            }
//          println("STSSimulateLLM:g_STSvYt=", STSvYt);
            return vYt;
}
```

```
////////////////////////////////////////////////////////////////
// simulate time series
////////////////////////////////////////////////////////////////
// Linear Trend Level Model
////////////////////////////////////////////////////////////////
STSSimulateLTL(const dSigma2Eta, const dSigma2Csi,
                                                const dSigma2Eps, const iYtSize) {
//          println("STSSimulateLTL:STSSimulateLTL()");
            decl iBurnIn=g_STSiBurnIn;
            decl vEta=sqrt(dSigma2Eta)*rann(1,iYtSize+iBurnIn);
            decl vCsi=sqrt(dSigma2Csi)*rann(1,iYtSize+iBurnIn);
            decl vEps=sqrt(dSigma2Eps)*rann(1,iYtSize+iBurnIn);
            decl vMu=zeros(1,iYtSize+iBurnIn);
            decl vBeta=zeros(1,iYtSize+iBurnIn);
            decl vYtAux=zeros(1,iYtSize+iBurnIn);
            decl vYt=zeros(1,iYtSize);
            decl i;
            vMu[0]=vEta[0];
            vBeta[0]=vCsi[0];
            vYtAux[0]=vMu[0]+vEps[0];
            for (i=1; i<iYtSize+iBurnIn; i++) {
                        vMu[i]=vMu[i-1]+vBeta[i-1]+vEta[i];
                        vBeta[i]=vBeta[i-1]+vCsi[i];
                        vYtAux[i]=vMu[i]+vEps[i];
            }
            for (i=0; i<iYtSize; i++) {
                        vYt[i]=vYtAux[i+iBurnIn];
            }
            return vYt;
}
```

```
////////////////////////////////////////////////////////////////
// simulate time series
////////////////////////////////////////////////////////////////
// Structural Basic Model
////////////////////////////////////////////////////////////////
STSSimulateSBM(const dSigma2Eta, const dSigma2Csi,
                                                const dSigma2Eps, const dSigma2Omg,
                                                const iSazo, const iYtSize) {
//          println("STSSimulateSBM:STSSimulateSBM()");
            decl iBurnIn=g_STSiBurnIn+iSazo;
            decl vEta=sqrt(dSigma2Eta)*rann(1,iYtSize+iBurnIn);
            decl vCsi=sqrt(dSigma2Csi)*rann(1,iYtSize+iBurnIn);
            decl vOmg=sqrt(dSigma2Omg)*rann(1,iYtSize+iBurnIn);
            decl vEps=sqrt(dSigma2Eps)*rann(1,iYtSize+iBurnIn);
            decl vMu=zeros(1,iYtSize+iBurnIn);
            decl vBeta=zeros(1,iYtSize+iBurnIn);
            decl vGamma=zeros(1,iYtSize+iBurnIn);
```

```
	decl vYtAux=zeros(1,iYtSize+iBurnIn);
	decl vYt=zeros(1,iYtSize);
	decl i, j;
	vMu[0]=vEta[0];
	vBeta[0]=vCsi[0];
	vGamma[0]=vOmg[0];
	for (i=1; i<iYtSize+iBurnIn; i++) {
		vMu[i]=vMu[i-1]+vBeta[i-1]+vEta[i];
		vBeta[i]=vBeta[i-1]+vCsi[i];
		vGamma[i]=vOmg[i];
		for (j=i-1; (j>=i-(iSazo-1))&&(j>=0); j--) {
			vGamma[i]=vGamma[i]-vGamma[j];
		}
		vYtAux[i]=vMu[i]+vGamma[i]+vEps[i];
	}
	for (i=0; i<iYtSize; i++) {
		vYt[i]=vYtAux[i+iBurnIn];
	}
//	decl sumvGamma;
//	for (i=0; i<iYtSize+iBurnIn; i++) {
//		sumvGamma=0.0;
//		for (j=i; (j>=i-iSazo+1)&&(j>=0); j--) {
//			sumvGamma=sumvGamma+vGamma[j];
//			print(vGamma[j], "\t");
//		}
//		println("\nsum\t", sumvGamma, "\tvOmg\t", vOmg[i]);
//	}
//	exit(0);
	return vYt;
}
////////////////////////////////////////////////////////////
// end
////////////////////////////////////////////////////////////
```

A.2 Biblioteca em Ox para Testar as Implementações dos Estimadores de Máxima Verossimilhança (EMV) e *Bootstrap*, e a Cobertura dos Intervalos de Confiança, para os Hiperparâmetros dos Modelos Estruturais

A.2.1 Modelo de Nível Local

```
////////////////////////////////////////////////////////////
#include <oxstd.h>
#include "STSBootPack.ox"
////////////////////////////////////////////////////////////
STSTestLLM(const STSiMC, const iSeed, const iYtSize,
	const dSigma2Eta, const dSigma2Eps) {
	println("STSTestLLM()");
	println("iYtSize\t", iYtSize);
	println("dSigma2Eta\t", dSigma2Eta);
	println("dSigma2Eps\t", dSigma2Eps);
	decl STSmModelLLM=<
		CMP_LEVEL, 1.0, 0, 0;
		CMP_IRREG, 1.0, 0, 0
	>;
	decl ii, STSvYt, STSmOmega, STSmRes;
	decl fSigma2Eta=zeros(1, STSiMC);
	decl fSigma2Eps=zeros(1, STSiMC);
	STSmModelLLM[0][1]=sqrt(dSigma2Eta);
	STSmModelLLM[1][1]=sqrt(dSigma2Eps);
	ranseed(iSeed);
	for (ii=0; ii<STSiMC; ii++) {
		STSvYt=STSSimulateLLM(dSigma2Eta, dSigma2Eps, iYtSize);
		STSmOmega=STSAdjust(STSvYt, STSmModelLLM);
		fSigma2Eta[ii]=STSmOmega[0][0];
		fSigma2Eps[ii]=STSmOmega[1][1];
		print(".");
//		println("STSTestLLM:STSvYt=", STSvYt);
//		println("STSTestLLM:STSmOmega=", STSmOmega);
//		exit(0);
	}
	println("");
//	println("STSTestLLM:E(Sigma2Eta)=", meanr(fSigma2Eta),
//		"STSTestLLM:V(Sigma2Eta)=", varr(fSigma2Eta),
//		"STSTestLLM:E(Sigma2Eps)=", meanr(fSigma2Eps),
```

```
//                      "STSTestLLM:V(Sigma2Eps)=", varr(fSigma2Eps));
        STSmRes=zeros(1,4);
        STSmRes[0]=meanr(fSigma2Eta);
        STSmRes[1]=sumsqrr(fSigma2Eta-dSigma2Eta)/STSiMC;
//      STSmRes[1]=varr(fSigma2Eta);
        STSmRes[2]=meanr(fSigma2Eps);
        STSmRes[3]=sumsqrr(fSigma2Eps-dSigma2Eps)/STSiMC;
//      STSmRes[3]=varr(fSigma2Eps);
        print("E(Sigma2Eta)\t", STSmRes[0]);
        println("\tEQM(Sigma2Eta)\t", STSmRes[1]);
//      print("V(Sigma2Eta)=", STSmRes[1]);
        print("E(Sigma2Eps)\t", STSmRes[2]);
        println("\tEQM(Sigma2Eps)\t", STSmRes[3]);
//      println("V(Sigma2Eps)=", STSmRes[3]);
}
```

```
STSBootCoverLLM(const STSiMC, const iSeed, const fAlpha,
        const iYtSize, const dSigma2Eta, const dSigma2Eps) {
        println("STSBootCoverLLM");
        println("fAlpha\t", fAlpha);
        println("iYtSize\t", iYtSize);
        println("dSigma2Eta\t", dSigma2Eta);
        println("dSigma2Eps\t", dSigma2Eps);
        decl STSmModelLLM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl ii, STSvYt, STSmOmega, STSmRes;
        decl fSigma2EtaEmv=zeros(1, STSiMC);
        decl fSigma2EpsEmv=zeros(1, STSiMC);
        decl fSigma2EtaBot=zeros(1, STSiMC);
        decl fSigma2EpsBot=zeros(1, STSiMC);
        decl fSigma2EtaLinf;
        decl fSigma2EtaLsup;
        decl fSigma2EpsLinf;
        decl fSigma2EpsLsup;
        decl fSigma2EtaCov=0;
        decl fSigma2EpsCov=0;
        STSmModelLLM[0][1]=sqrt(dSigma2Eta);
        STSmModelLLM[1][1]=sqrt(dSigma2Eps);
        ranseed(iSeed);
        for (ii=0; ii<STSiMC; ii++) {
                print(ii+1, " ");
                STSvYt=STSSimulateLLM(dSigma2Eta, dSigma2Eps, iYtSize);
                STSmOmega=STSAdjust(STSvYt, STSmModelLLM);
                STSmRes=STSBootCI(STSvYt, STSmModelLLM);
                fSigma2EtaEmv[ii]=STSmOmega[0][0];
                fSigma2EpsEmv[ii]=STSmOmega[1][1];
                fSigma2EtaBot[ii]=meanr(STSmRes[0][:]);
                fSigma2EpsBot[ii]=meanr(STSmRes[1][:]);
                fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
                fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
                fSigma2EpsLinf=STSQuantile(STSmRes[1][:],(1-fAlpha)/2);
                fSigma2EpsLsup=STSQuantile(STSmRes[1][:],(1+fAlpha)/2);
                if ((dSigma2Eta>=fSigma2EtaLinf)&&(dSigma2Eta<=fSigma2EtaLsup))
                        fSigma2EtaCov++;
                if ((dSigma2Eps>=fSigma2EpsLinf)&&(dSigma2Eps<=fSigma2EpsLsup))
                        fSigma2EpsCov++;
        }
        println("Eemv(Sigma2Eta)\t", meanr(fSigma2EtaEmv));
        println("EQMemv(Sigma2Eta)\t", sumsqrr(fSigma2EtaEmv-dSigma2Eta)/STSiMC);
        println("Ebot(Sigma2Eta)\t", meanr(fSigma2EtaBot));
        println("EQMbot(Sigma2Eta)\t", sumsqrr(fSigma2EtaBot-dSigma2Eta)/STSiMC);
        println("Coverage(Sigma2Eta)\n\t", fSigma2EtaCov/STSiMC);
//
        println("Eemv(Sigma2Eps)\t", meanr(fSigma2EpsEmv));
        println("EQMemv(Sigma2Eps)\t", sumsqrr(fSigma2EpsEmv-dSigma2Eps)/STSiMC);
        println("Ebot(Sigma2Eps)\t", meanr(fSigma2EpsBot));
        println("EQMbot(Sigma2Eps)\t", sumsqrr(fSigma2EpsBot-dSigma2Eps)/STSiMC);
        println("Coverage(Sigma2Eps)\n\t", fSigma2EpsCov/STSiMC);
}
```

A.2.2 Modelo com Tendência Linear Local

```
STSTestLTL(const STSiMC, const iSeed, const iYtSize,
        const dSigma2Eta, const dSigma2Csi, const dSigma2Eps) {
```

```
        println("STSTestLTL()");
        println("iYtSize\t", iYtSize);
        println("dSigma2Eta\t", dSigma2Eta);
        println("dSigma2Csi\t", dSigma2Csi);
        println("dSigma2Eps\t", dSigma2Eps);
        decl STSmModelLTL=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl ii, STSvYt, STSmOmega, STSmRes;
        decl fSigma2Eta=zeros(1, STSiMC);
        decl fSigma2Csi=zeros(1, STSiMC);
        decl fSigma2Eps=zeros(1, STSiMC);
        STSmModelLTL[0][1]=sqrt(dSigma2Eta);
        STSmModelLTL[1][1]=sqrt(dSigma2Csi);
        STSmModelLTL[2][1]=sqrt(dSigma2Eps);
//      GetSsfStsm(STSmModelLTL,
//              &g_STSmPhi, &g_STSmOmega, &g_STSmSigma);
//      println("g_STSmPhi=", g_STSmPhi,
//              "g_STSmOmega=", g_STSmOmega,
//              "g_STSmSigma=", g_STSmSigma);
//      exit(0);
        ranseed(iSeed);
        for (ii=0; ii<STSiMC; ii++) {
                STSvYt=STSSimulateLTL(dSigma2Eta, dSigma2Csi,
                        dSigma2Eps, iYtSize);
                STSmOmega=STSAdjust(STSvYt, STSmModelLTL);
                fSigma2Eta[ii]=STSmOmega[0][0];
                fSigma2Csi[ii]=STSmOmega[1][1];
                fSigma2Eps[ii]=STSmOmega[2][2];
                print(".");
//              println("STSTestLTL:STSvYt=", STSvYt);
//              println("STSTestLTL:STSmOmega=", STSmOmega);
//              exit(0);
        }
        println("");
//      print("STSTestLTL:E(Sigma2Eta)=", meanr(fSigma2Eta),
//              "STSTestLTL:V(Sigma2Eta)=", varr(fSigma2Eta),
//              "STSTestLTL:E(Sigma2Csi)=", meanr(fSigma2Csi),
//              "STSTestLTL:V(Sigma2Csi)=", varr(fSigma2Csi),
//              "STSTestLTL:E(Sigma2Eps)=", meanr(fSigma2Eps),
//              "STSTestLTL:V(Sigma2Eps)=", varr(fSigma2Eps));
        STSmRes=zeros(1,6);
        STSmRes[0]=meanr(fSigma2Eta);
        STSmRes[1]=sumsqrr(fSigma2Eta-dSigma2Eta)/STSiMC;
//      STSmRes[1]=varr(fSigma2Eta);
        STSmRes[2]=meanr(fSigma2Csi);
        STSmRes[3]=sumsqrr(fSigma2Csi-dSigma2Csi)/STSiMC;
//      STSmRes[3]=varr(fSigma2Csi);
        STSmRes[4]=meanr(fSigma2Eps);
        STSmRes[5]=sumsqrr(fSigma2Eps-dSigma2Eps)/STSiMC;
//      STSmRes[5]=varr(fSigma2Eps);
        print("E(Sigma2Eta)\t", STSmRes[0]);
        println("\tEQM(Sigma2Eta)\t", STSmRes[1]);
//      println("V(Sigma2Eta)=", STSmRes[1]);
        print("E(Sigma2Csi)\t", STSmRes[2]);
        println("\tEQM(Sigma2Csi)\t", STSmRes[3]);
//      println("V(Sigma2Csi)=", STSmRes[3]);
        print("E(Sigma2Eps)\t", STSmRes[4]);
        println("\tEQM(Sigma2Eps)\t", STSmRes[5]);
//      println("V(Sigma2Eps)=", STSmRes[5]);
}
```

```
STSBootCoverLTL(const STSiMC, const iSeed, const fAlpha,
        const iYtSize, const dSigma2Eta, const dSigma2Csi,
        const dSigma2Eps) {
        println("STSBootCoverLTL");
        println("fAlpha\t", fAlpha);
        println("iYtSize\t", iYtSize);
        println("dSigma2Eta\t", dSigma2Eta);
        println("dSigma2Csi\t", dSigma2Csi);
        println("dSigma2Eps\t", dSigma2Eps);
        decl STSmModelLTL=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl ii, STSvYt, STSmOmega, STSmRes;
```

```
decl fSigma2EtaEmv=zeros(1, STSiMC);
decl fSigma2CsiEmv=zeros(1, STSiMC);
decl fSigma2EpsEmv=zeros(1, STSiMC);
decl fSigma2EtaBot=zeros(1, STSiMC);
decl fSigma2CsiBot=zeros(1, STSiMC);
decl fSigma2EpsBot=zeros(1, STSiMC);
decl fSigma2EtaLinf;
decl fSigma2EtaLsup;
decl fSigma2CsiLinf;
decl fSigma2CsiLsup;
decl fSigma2EpsLinf;
decl fSigma2EpsLsup;
decl fSigma2EtaCov=0;
decl fSigma2CsiCov=0;
decl fSigma2EpsCov=0;
STSmModelLTL[0][1]=sqrt(dSigma2Eta);
STSmModelLTL[1][1]=sqrt(dSigma2Csi);
STSmModelLTL[2][1]=sqrt(dSigma2Eps);
ranseed(iSeed);
for (ii=0; ii<STSiMC; ii++) {
        print(ii+1, " ");
        STSvYt=STSSimulateLTL(dSigma2Eta, dSigma2Csi,
                dSigma2Eps, iYtSize);
        STSmOmega=STSAdjust(STSvYt, STSmModelLTL);
        STSmRes=STSBootCI(STSvYt, STSmModelLTL);
        fSigma2EtaEmv[ii]=STSmOmega[0][0];
        fSigma2CsiEmv[ii]=STSmOmega[1][1];
        fSigma2EpsEmv[ii]=STSmOmega[2][2];
        fSigma2EtaBot[ii]=meanr(STSmRes[0][:]);
        fSigma2CsiBot[ii]=meanr(STSmRes[1][:]);
        fSigma2EpsBot[ii]=meanr(STSmRes[2][:]);
        fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
        fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
        fSigma2CsiLinf=STSQuantile(STSmRes[1][:],(1-fAlpha)/2);
        fSigma2CsiLsup=STSQuantile(STSmRes[1][:],(1+fAlpha)/2);
        fSigma2EpsLinf=STSQuantile(STSmRes[2][:],(1-fAlpha)/2);
        fSigma2EpsLsup=STSQuantile(STSmRes[2][:],(1+fAlpha)/2);
        if ((dSigma2Eta>=fSigma2EtaLinf)&&(dSigma2Eta<=fSigma2EtaLsup))
                fSigma2EtaCov++;
        if ((dSigma2Csi>=fSigma2CsiLinf)&&(dSigma2Csi<=fSigma2CsiLsup))
                fSigma2CsiCov++;
        if ((dSigma2Eps>=fSigma2EpsLinf)&&(dSigma2Eps<=fSigma2EpsLsup))
                fSigma2EpsCov++;
    }
    println("Eemv(Sigma2Eta)\t", meanr(fSigma2EtaEmv));
    println("EQMemv(Sigma2Eta)\t", sumsqrr(fSigma2EtaEmv-dSigma2Eta)/STSiMC);
    println("Ebot(Sigma2Eta)\t", meanr(fSigma2EtaBot));
    println("EQMbot(Sigma2Eta)\t", sumsqrr(fSigma2EtaBot-dSigma2Eta)/STSiMC);
    println("Coverage(Sigma2Eta)\n\t", fSigma2EtaCov/STSiMC);
//
    println("Eemv(Sigma2Csi)\t", meanr(fSigma2CsiEmv));
    println("EQMemv(Sigma2Csi)\t", sumsqrr(fSigma2CsiEmv-dSigma2Csi)/STSiMC);
    println("Ebot(Sigma2Csi)\t", meanr(fSigma2CsiBot));
    println("EQMbot(Sigma2Csi)\t", sumsqrr(fSigma2CsiBot-dSigma2Csi)/STSiMC);
    println("Coverage(Sigma2Csi)\n\t", fSigma2CsiCov/STSiMC);
//
    println("Eemv(Sigma2Eps)\t", meanr(fSigma2EpsEmv));
    println("EQMemv(Sigma2Eps)\t", sumsqrr(fSigma2EpsEmv-dSigma2Eps)/STSiMC);
    println("Ebot(Sigma2Eps)\t", meanr(fSigma2EpsBot));
    println("EQMbot(Sigma2Eps)\t", sumsqrr(fSigma2EpsBot-dSigma2Eps)/STSiMC);
    println("Coverage(Sigma2Eps)\n\t", fSigma2EpsCov/STSiMC);
}
```

A.2.3 Modelo Estrutural Básico

```
STSTestSBM(const STSiMC, const iSeed, const iYtSize,
        const dSigma2Eta, const dSigma2Csi, const dSigma2Omg,
        const iSazo, const dSigma2Eps) {
    println("STSTestSBM()");
    println("iYtSize\t", iYtSize);
    println("dSigma2Eta\t", dSigma2Eta);
    println("dSigma2Csi\t", dSigma2Csi);
    println("dSigma2Omg\t", dSigma2Omg);
    println("iSazo\t", iSazo);
    println("dSigma2Eps\t", dSigma2Eps);
    decl STSmModelSBM=<
            CMP_LEVEL, 1.0, 0, 0;
            CMP_SLOPE, 1.0, 0, 0;
```

```
                        CMP_SEAS_DUMMY, 1.0, 1, 0;
                        CMP_IRREG, 1.0, 0, 0
        >;
        decl ii, ij, STSvYt, STSmOmega, STSmRes;
        decl fSigma2Eta=zeros(1, STSiMC);
        decl fSigma2Csi=zeros(1, STSiMC);
        decl fSigma2Omg=zeros(1, STSiMC);
        decl fSigma2Eps=zeros(1, STSiMC);
        STSmModelSBM[0][1]=sqrt(dSigma2Eta);
        STSmModelSBM[1][1]=sqrt(dSigma2Csi);
        STSmModelSBM[2][1]=sqrt(dSigma2Omg);
        STSmModelSBM[2][2]=iSazo;
        STSmModelSBM[3][1]=sqrt(dSigma2Eps);
//      GetSsfStsm(STSmModelSBM,
//                      &g_STSmPhi, &g_STSmOmega, &g_STSmSigma);
//      println("g_STSmPhi=", g_STSmPhi,
//                      "g_STSmOmega=", g_STSmOmega,
//                      "g_STSmSigma=", g_STSmSigma);
//      exit(0);
        ranseed(iSeed);
        for (ii=0; ii<STSiMC; ii++) {
                        STSvYt=STSSimulateSBM(dSigma2Eta, dSigma2Csi,
                                        dSigma2Eps, dSigma2Omg, iSazo, iYtSize);
                        STSmOmega=STSAdjust(STSvYt, STSmModelSBM);
                        ij=rows(STSmOmega);
                        fSigma2Eta[ii]=STSmOmega[0][0];
                        fSigma2Csi[ii]=STSmOmega[1][1];
                        fSigma2Omg[ii]=STSmOmega[2][2];
                        fSigma2Eps[ii]=STSmOmega[ij-1][ij-1];
                        print(".");
//                      println("STSTestSBM:STSvYt=");
//                      for (ij=0; ij<iYtSize; ij++) println(STSvYt[ij]);
//                      println("STSTestSBM:STSmOmega=", STSmOmega);
//                      exit(0);
        }
        println("");
//      print("E(Sigma2Eta)=", meanr(fSigma2Eta),
//                      "V(Sigma2Eta)=", varr(fSigma2Eta),
//                      "E(Sigma2Csi)=", meanr(fSigma2Csi),
//                      "V(Sigma2Csi)=", varr(fSigma2Csi),
//                      "E(Sigma2Eps)=", meanr(fSigma2Eps),
//                      "V(Sigma2Eps)=", varr(fSigma2Eps),
//                      "E(Sigma2Omg)=", meanr(fSigma2Omg),
//                      "V(Sigma2Omg)=", varr(fSigma2Omg));
        STSmRes=zeros(1,8);
        STSmRes[0]=meanr(fSigma2Eta);
        STSmRes[1]=sumsqrr(fSigma2Eta-dSigma2Eta)/STSiMC;
//      STSmRes[1]=varr(fSigma2Eta);
        STSmRes[2]=meanr(fSigma2Csi);
        STSmRes[3]=sumsqrr(fSigma2Csi-dSigma2Csi)/STSiMC;
//      STSmRes[3]=varr(fSigma2Csi);
        STSmRes[4]=meanr(fSigma2Omg);
        STSmRes[5]=sumsqrr(fSigma2Omg-dSigma2Omg)/STSiMC;
//      STSmRes[5]=varr(fSigma2Omg);
        STSmRes[6]=meanr(fSigma2Eps);
        STSmRes[7]=sumsqrr(fSigma2Eps-dSigma2Eps)/STSiMC;
//      STSmRes[7]=varr(fSigma2Eps);
        print("E(Sigma2Eta)\t", STSmRes[0]);
        println("\tEQM(Sigma2Eta)\t", STSmRes[1]);
//      println("V(Sigma2Eta)=", STSmRes[1]);
        print("E(Sigma2Csi)\t", STSmRes[2]);
        println("\tEQM(Sigma2Csi)\t", STSmRes[3]);
//      println("V(Sigma2Csi)=", STSmRes[3]);
        print("E(Sigma2Omg)\t", STSmRes[4]);
        println("\tEQM(Sigma2Omg)\t", STSmRes[5]);
//      println("V(Sigma2Omg)=", STSmRes[5]);
        print("E(Sigma2Eps)\t", STSmRes[6]);
        println("\tEQM(Sigma2Eps)\t", STSmRes[7]);
//      println("V(Sigma2Eps)=", STSmRes[7]);
}

STSBootCoverSBM(const STSiMC, const iSeed, const fAlpha,
        const iYtSize, const dSigma2Eta, const dSigma2Csi,
        const dSigma2Omg, const iSazo, const dSigma2Eps) {
        println("STSBootCoverSBM");
        println("fAlpha\t", fAlpha);
        println("iYtSize\t", iYtSize);
        println("dSigma2Eta\t", dSigma2Eta);
        println("dSigma2Csi\t", dSigma2Csi);
```

```
println("dSigma2Omg\t", dSigma2Omg);
println("iSazo\t", iSazo);
println("dSigma2Eps\t", dSigma2Eps);
decl STSmModelSBM=<
        CMP_LEVEL, 1.0, 0, 0;
        CMP_SLOPE, 1.0, 0, 0;
        CMP_SEAS_DUMMY, 1.0, 4, 0;
        CMP_IRREG, 1.0, 0, 0
>;
decl ii, ij, STSvYt, STSmOmega, STSmRes;
decl fSigma2EtaEmv=zeros(1, STSiMC);
decl fSigma2CsiEmv=zeros(1, STSiMC);
decl fSigma2OmgEmv=zeros(1, STSiMC);
decl fSigma2EpsEmv=zeros(1, STSiMC);
decl fSigma2EtaBot=zeros(1, STSiMC);
decl fSigma2CsiBot=zeros(1, STSiMC);
decl fSigma2OmgBot=zeros(1, STSiMC);
decl fSigma2EpsBot=zeros(1, STSiMC);
decl fSigma2EtaLinf;
decl fSigma2EtaLsup;
decl fSigma2CsiLinf;
decl fSigma2CsiLsup;
decl fSigma2OmgLinf;
decl fSigma2OmgLsup;
decl fSigma2EpsLinf;
decl fSigma2EpsLsup;
decl fSigma2EtaCov=0;
decl fSigma2CsiCov=0;
decl fSigma2OmgCov=0;
decl fSigma2EpsCov=0;
STSmModelSBM[0][1]=sqrt(dSigma2Eta);
STSmModelSBM[1][1]=sqrt(dSigma2Csi);
STSmModelSBM[2][1]=sqrt(dSigma2Omg);
STSmModelSBM[2][2]=iSazo;
STSmModelSBM[3][1]=sqrt(dSigma2Eps);
ranseed(iSeed);
for (ii=0; ii<STSiMC; ii++) {
        print(ii+1, " ");
        STSvYt=STSSimulateSBM(dSigma2Eta, dSigma2Csi,
                dSigma2Eps, dSigma2Omg, iSazo, iYtSize);
        STSmOmega=STSAdjust(STSvYt, STSmModelSBM);
        STSmRes=STSBootCI(STSvYt, STSmModelSBM);
        ij=rows(STSmOmega);
        fSigma2EtaEmv[ii]=STSmOmega[0][0];
        fSigma2CsiEmv[ii]=STSmOmega[1][1];
        fSigma2OmgEmv[ii]=STSmOmega[2][2];
        fSigma2EpsEmv[ii]=STSmOmega[ij-1][ij-1];
        fSigma2EtaBot[ii]=meanr(STSmRes[0][:]);
        fSigma2CsiBot[ii]=meanr(STSmRes[1][:]);
        fSigma2OmgBot[ii]=meanr(STSmRes[2][:]);
        fSigma2EpsBot[ii]=meanr(STSmRes[3][:]);
        fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
        fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
        fSigma2CsiLinf=STSQuantile(STSmRes[1][:],(1-fAlpha)/2);
        fSigma2CsiLsup=STSQuantile(STSmRes[1][:],(1+fAlpha)/2);
        fSigma2OmgLinf=STSQuantile(STSmRes[2][:],(1-fAlpha)/2);
        fSigma2OmgLsup=STSQuantile(STSmRes[2][:],(1+fAlpha)/2);
        fSigma2EpsLinf=STSQuantile(STSmRes[3][:],(1-fAlpha)/2);
        fSigma2EpsLsup=STSQuantile(STSmRes[3][:],(1+fAlpha)/2);
        if ((dSigma2Eta>=fSigma2EtaLinf)&&(dSigma2Eta<=fSigma2EtaLsup))
                fSigma2EtaCov++;
        if ((dSigma2Csi>=fSigma2CsiLinf)&&(dSigma2Csi<=fSigma2CsiLsup))
                fSigma2CsiCov++;
        if ((dSigma2Omg>=fSigma2OmgLinf)&&(dSigma2Omg<=fSigma2OmgLsup))
                fSigma2OmgCov++;
        if ((dSigma2Eps>=fSigma2EpsLinf)&&(dSigma2Eps<=fSigma2EpsLsup))
                fSigma2EpsCov++;
}
println("Eemv(Sigma2Eta)\t", meanr(fSigma2EtaEmv));
println("EQMemv(Sigma2Eta)\t", sumsqrr(fSigma2EtaEmv-dSigma2Eta)/STSiMC);
println("Ebot(Sigma2Eta)\t", meanr(fSigma2EtaBot));
println("EQMbot(Sigma2Eta)\t", sumsqrr(fSigma2EtaBot-dSigma2Eta)/STSiMC);
println("Coverage(Sigma2Eta)\n\t", fSigma2EtaCov/STSiMC);
//
println("Eemv(Sigma2Csi)\t", meanr(fSigma2CsiEmv));
println("EQMemv(Sigma2Csi)\t", sumsqrr(fSigma2CsiEmv-dSigma2Csi)/STSiMC);
println("Ebot(Sigma2Csi)\t", meanr(fSigma2CsiBot));
println("EQMbot(Sigma2Csi)\t", sumsqrr(fSigma2CsiBot-dSigma2Csi)/STSiMC);
println("Coverage(Sigma2Csi)\n\t", fSigma2CsiCov/STSiMC);
//
```

```
        println("Eemv(Sigma2Omg)\t", meanr(fSigma2OmgEmv));
        println("EQMemv(Sigma2Omg)\t", sumsqrr(fSigma2OmgEmv-dSigma2Omg)/STSiMC);
        println("Ebot(Sigma2Omg)\t", meanr(fSigma2OmgBot));
        println("EQMbot(Sigma2Omg)\t", sumsqrr(fSigma2OmgBot-dSigma2Omg)/STSiMC);
        println("Coverage(Sigma2Omg)\n\t", fSigma2OmgCov/STSiMC);
//
        println("Eemv(Sigma2Eps)\t", meanr(fSigma2EpsEmv));
        println("EQMemv(Sigma2Eps)\t", sumsqrr(fSigma2EpsEmv-dSigma2Eps)/STSiMC);
        println("Ebot(Sigma2Eps)\t", meanr(fSigma2EpsBot));
        println("EQMbot(Sigma2Eps)\t", sumsqrr(fSigma2EpsBot-dSigma2Eps)/STSiMC);
        println("Coverage(Sigma2Eps)\n\t", fSigma2EpsCov/STSiMC);
}
```

A.3 Programa em Ox para Gerar Tabelas 3.1, 3.3 e 3.5 e Tabelas B.1, B.2 e B.3

```
//////////////////////////////////////////////////////////////
main() {
//////////////////////////////////////////////////////////////
//      tabela iter. BFGS x burn-in x replic. MC x tamanho
        STSTestLLM(STSiMC, iSeed, iYtSize, dSigma2Eta, dSigma2Eps);
        g_STSiBurnIn=100;
        g_STSiBFGS=50;
        STSTestLLM( 500, 26480, 50, 0.50, 1.00);
//      tabela EMV - bootstrap
        STSBootCoverLLM(STSiMC, iSeed, fAlpha, iYtSize,     dSigma2Eta, dSigma2Eps);
        g_STSiBFGS=50;
        g_STSiBurnIn=100;
        STSBootCoverLLM(500, 26480, 0.95, 50, 0.50, 1.00);
//      tabela iter. BFGS x burn-in x replic. MC x tamanho
        STSTestLTL(STSiMC, iSeed, iYtSize, dSigma2Eta, dSigma2Csi, dSigma2Eps);
        g_STSiBFGS=50;
        g_STSiBurnIn=100;
        STSTestLTL( 500, 26480, 50, 0.50, 0.10, 1.00);
//      tabela EMV - bootstrap
        STSBootCoverLTL(STSiMC, iSeed, fAlpha, iYtSize,
        dSigma2Eta, dSigma2Csi, dSigma2Eps);
        g_STSiBFGS=50;
        g_STSiBurnIn=100;
        STSBootCoverLTL(500, 26480, 0.95, 50, 0.50, 0. 10, 1.00);
//      tabela iter. BFGS x burn-in x replic. MC x tamanho
        STSTestSBM(STSiMC, iSeed, iYtSize, dSigma2Eta, dSigma2Csi, dSigma2Omg, dSigma2Eps);
        g_STSiBFGS=50;
        g_STSiBurnIn=100;
        STSTestSBM( 500, 26480, 50, 0.50, 0. 01, 0. 10, 4, 1.00);
//      tabela EMV - bootstrap
        STSBootCoverSBM(STSiMC, iSeed, fAlpha, iYtSize, dSigma2Eta, dSigma2Csi,
                dSigma2Omg, iSazo, dSigma2Eps)
        g_STSiBFGS=50;
        g_STSiBurnIn=100;
        STSBootCoverSBM(500, 24680, 0.95, 50, 0.50, 0.01, 0.10, 4, 1.00);
        exit(0);
}
//////////////////////////////////////////////////////////////
```

APÊNDICE B

Tabela B.1: Resultado de simulação para MNL variando o número de iterações BFGS e de replicações MC (*burn-in*=100)

Tamanho da série	Nº de Iterações BFGS	Nº de Replicações MC	$\sigma_\eta^2 = 0{,}50$		$\sigma_\varepsilon^2 = 1{,}00$	
			Média	EQM	Média	EQM
50	50	100	0,4872	0,0833	1,0109	0,1185
		500	0,5109	0,0868	0,9911	0,1082
		1.000	0,5133	0,0873	0,9871	0,1065
	200	100	0,4872	0,0833	1,0109	0,1185
		500	0,5109	0,0868	0,9911	0,1082
		1.000	0,5133	0,0873	0,9871	0,1065
	1.000	100	0,4872	0,0833	1,0109	0,1185
		500	0,5109	0,0868	0,9911	0,1082
		1.000	0,5133	0,0873	0,9871	0,1065

Tabela B.2: Resultado de simulação para MTL variando o número de iterações BFGS e de replicações MC (*burn-in*=100)

Tamanho da série	Nº de Iterações BFGS	Nº de Replicações MC	$\sigma_\eta^2 = 0{,}50$		$\sigma_\xi^2 = 0{,}10$		$\sigma_\varepsilon^2 = 1{,}00$	
			Média	EQM	Média	EQM	Média	EQM
50	50	100	0,5344	0,3318	0,0883	0,0051	0,9534	0,1621
		500	0,5506	0,3763	0,0951	0,0060	0,9763	0,1517
		1.000	0,5701	0,3930	0,0933	0,0057	0,9648	0,1546
	200	100	0,5344	0,3318	0,0883	0,0051	0,9534	0,1621
		500	0,5506	0,3763	0,0951	0,0060	0,9763	0,1517
		1.000	0,5701	0,3930	0,0933	0,0057	0,9648	0,1546
	1.000	100	0,5344	0,3318	0,0883	0,0051	0,9534	0,1621
		500	0,5506	0,3763	0,0951	0,0060	0,9763	0,1517
		1.000	0,5701	0,3930	0,0933	0,0057	0,9648	0,1546

Tabela B.3: Resultado de simulação para MEB variando o número de iterações BFGS e de replicações MC (*burn-in*=100)

Tamanho da série	Nº de Iterações BFGS	Nº de Replicações MC	$\sigma_\eta^2 = 0{,}50$		$\sigma_\xi^2 = 0{,}01$		$\sigma_\omega^2 = 0{,}10$		$\sigma_\varepsilon^2 = 1{,}00$	
			Média	EQM	Média	EQM	Média	EQM	Média	EQM
50	50	100	0,4094	0,1560	0,0137	0,0004	0,0910	0,0093	0,9941	0,1615
		500	0,4541	0,1601	0,0113	0,0003	0,1001	0,0078	0,9936	0,2186
		1.000	0,4649	0,1635	0,0114	0,0003	0,1030	0,0089	0,9860	0,2286
	200	100	0,4094	0,1560	0,0137	0,0004	0,0910	0,0093	0,9941	0,1615
		500	0,4541	0,1601	0,0113	0,0003	0,1001	0,0078	0,9936	0,2187
		1.000	0,4649	0,1635	0,0114	0,0003	0,1030	0,0089	0,9860	0,2286
	1.000	100	0,4094	0,1560	0,0137	0,0004	0,0910	0,0093	0,9941	0,1615
		500	0,4541	0,1601	0,0113	0,0003	0,1001	0,0078	0,9936	0,2187
		1.000	0,4649	0,1635	0,0114	0,0003	0,1030	0,0089	0,9860	0,2286

Apêndice C

C.1 Programa Em Ox para Leitura da Série Real IPCA (%) e Ajuste de Um Modelo Estrutural Básico

```
////////////////////////////////////////////////////////////////////
// Purpose:
//          This program adjusts models for
//          structural time series
//          Structural Basic Model
////////////////////////////////////////////////////////////////////
#include <oxstd.h>
#include "STSBootPack.ox"
//////////////////////////////////////////////////////////////
main() {
//////////////////////////////////////////////////////////////
        decl STSvYt, STSmOmega, STSmRes;
        decl fSigma2EtaEmv;
        decl fSigma2CsiEmv;
        decl fSigma2OmgEmv;
        decl fSigma2EpsEmv;
        decl fSigma2EtaBot;
        decl fSigma2CsiBot;
        decl fSigma2OmgBot;
        decl fSigma2EpsBot;
        decl fSigma2EtaLinf;
        decl fSigma2EtaLsup;
        decl fSigma2CsiLinf;
        decl fSigma2CsiLsup;
        decl fSigma2OmgLinf;
        decl fSigma2OmgLsup;
        decl fSigma2EpsLinf;
        decl fSigma2EpsLsup;
        decl fAlpha=0.95;
        decl STSmModelLLM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelLTL=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelSBM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_SEAS_DUMMY, 1.0, 12, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
//      ranseed(1357924680);
        ranseed(0);
        g_STSiBFGS=50;
        g_STSiBoot=1000;
//////////////////////////////////////////////////////////////
//      Structural Basic Model
//////////////////////////////////////////////////////////////
        STSvYt=loadmat("ipca.dat");
        STSvYt=STSvYt';
        println("Data:\n", STSvYt);
        println("STSAdjust:STSAdjust()");
        STSmOmega=STSAdjust(STSvYt, STSmModelSBM);
        STSmRes=STSBootCI(STSvYt, STSmModelSBM);
        fSigma2EtaEmv=STSmOmega[0][0];
        fSigma2CsiEmv=STSmOmega[1][1];
        fSigma2OmgEmv=STSmOmega[2][2];
        fSigma2EpsEmv=STSmOmega[rows(STSmOmega)-1][rows(STSmOmega)-1];
        fSigma2EtaBot=meanr(STSmRes[0][:])[0];
        fSigma2CsiBot=meanr(STSmRes[1][:])[0];
        fSigma2OmgBot=meanr(STSmRes[2][:])[0];
        fSigma2EpsBot=meanr(STSmRes[rows(STSmRes)-1][:])[0];
        fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
        fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
        fSigma2CsiLinf=STSQuantile(STSmRes[1][:],(1-fAlpha)/2);
        fSigma2CsiLsup=STSQuantile(STSmRes[1][:],(1+fAlpha)/2);
        fSigma2OmgLinf=STSQuantile(STSmRes[2][:],(1-fAlpha)/2);
        fSigma2OmgLsup=STSQuantile(STSmRes[2][:],(1+fAlpha)/2);
        fSigma2EpsLinf=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1-fAlpha)/2);
        fSigma2EpsLsup=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1+fAlpha)/2);
        println("Adjust:\nParameter\tEstimate\tBootEstimate\tLInf\tLSup");
        println("Sigma2Eta\t",
                fSigma2EtaEmv, "\t", fSigma2EtaBot, "\t",
                fSigma2EtaLinf, "\t", fSigma2EtaLsup);
        println("Sigma2Csi\t",
                fSigma2CsiEmv, "\t", fSigma2CsiBot, "\t",
                fSigma2CsiLinf, "\t", fSigma2CsiLsup);
        println("Sigma2Omg\t",
                fSigma2OmgEmv, "\t", fSigma2OmgBot, "\t",
                fSigma2OmgLinf, "\t", fSigma2OmgLsup);
        println("Sigma2Eps\t",
                fSigma2EpsEmv, "\t", fSigma2EpsBot, "\t",
                fSigma2EpsLinf, "\t", fSigma2EpsLsup);
        exit(0);
}
```

C.2 Programa Em Ox para Leitura da Série Real IPCA (%) e Ajuste de Um Modelo com Tendência Local

```
////////////////////////////////////////////////////////////////////
// Purpose:
//          This program adjusts models for
//          structural time series
//          Linear Trend Level Model
////////////////////////////////////////////////////////////////////
#include <oxstd.h>
#include "STSBootPack.ox"
/////////////////////////////////////////////////////////////
main() {
/////////////////////////////////////////////////////////////
        decl STSvYt, STSmOmega, STSmRes;
        decl fSigma2EtaEmv;
        decl fSigma2CsiEmv;
        decl fSigma2OmgEmv;
        decl fSigma2EpsEmv;
        decl fSigma2EtaBot;
        decl fSigma2CsiBot;
        decl fSigma2OmgBot;
        decl fSigma2EpsBot;
        decl fSigma2EtaLinf;
        decl fSigma2EtaLsup;
        decl fSigma2CsiLinf;
        decl fSigma2CsiLsup;
        decl fSigma2OmgLinf;
        decl fSigma2OmgLsup;
        decl fSigma2EpsLinf;
        decl fSigma2EpsLsup;
        decl fAlpha=0.95;
        decl STSmModelLLM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelLTL=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelSBM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_SEAS_DUMMY, 1.0, 12, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
//      ranseed(1357924680);
        ranseed(0);
        g_STSiBFGS=50;
        g_STSiBoot=10;
/////////////////////////////////////////////////////////////
//      Linear Trend Level Model
/////////////////////////////////////////////////////////////
        STSvYt=loadmat("ipca.dat");
        STSvYt=STSvYt';
        println("Data:\n", STSvYt);
        println("STSAdjust:STSAdjust()");
        STSmOmega=STSAdjust(STSvYt, STSmModelLTL);
        STSmRes=STSBootCI(STSvYt, STSmModelLTL);
        fSigma2EtaEmv=STSmOmega[0][0];
        fSigma2CsiEmv=STSmOmega[1][1];
        fSigma2EpsEmv=STSmOmega[rows(STSmOmega)-1][rows(STSmOmega)-1];
        fSigma2EtaBot=meanr(STSmRes[0][:])[0];
        fSigma2CsiBot=meanr(STSmRes[1][:])[0];
        fSigma2EpsBot=meanr(STSmRes[rows(STSmRes)-1][:])[0];
        fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
        fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
        fSigma2CsiLinf=STSQuantile(STSmRes[1][:],(1-fAlpha)/2);
        fSigma2CsiLsup=STSQuantile(STSmRes[1][:],(1+fAlpha)/2);
        fSigma2EpsLinf=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1-fAlpha)/2);
        fSigma2EpsLsup=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1+fAlpha)/2);
        println("Adjust:\nParameter\tEstimate\tBootEstimate\tLInf\tLSup");
        println("Sigma2Eta\t",
                fSigma2EtaEmv, "\t", fSigma2EtaBot, "\t",
                fSigma2EtaLinf, "\t", fSigma2EtaLsup);
        println("Sigma2Csi\t",
                fSigma2CsiEmv, "\t", fSigma2CsiBot, "\t",
                fSigma2CsiLinf, "\t", fSigma2CsiLsup);
        println("Sigma2Eps\t",
                fSigma2EpsEmv, "\t", fSigma2EpsBot, "\t",
                fSigma2EpsLinf, "\t", fSigma2EpsLsup);
        exit(0);
}
```

C.3 Programa Em Ox para Leitura da Série Real IPCA (%) e Ajuste de um Modelo de Nível Local

```
////////////////////////////////////////////////////////////////////
// Purpose:
//          This program adjusts models for
//          structural time series
//          Local Level Model
////////////////////////////////////////////////////////////////////
#include <oxstd.h>
#include "STSBootPack.ox"
//////////////////////////////////////////////////////////////
main() {
//////////////////////////////////////////////////////////////
        decl STSvYt, STSmOmega, STSmRes;
        decl fSigma2EtaEmv;
        decl fSigma2CsiEmv;
        decl fSigma2OmgEmv;
        decl fSigma2EpsEmv;
        decl fSigma2EtaBot;
        decl fSigma2CsiBot;
        decl fSigma2OmgBot;
        decl fSigma2EpsBot;
        decl fSigma2EtaLinf;
        decl fSigma2EtaLsup;
        decl fSigma2CsiLinf;
        decl fSigma2CsiLsup;
        decl fSigma2OmgLinf;
        decl fSigma2OmgLsup;
        decl fSigma2EpsLinf;
        decl fSigma2EpsLsup;
        decl fAlpha=0.95;
        decl STSmModelLLM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelLTL=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
        decl STSmModelSBM=<
                CMP_LEVEL, 1.0, 0, 0;
                CMP_SLOPE, 1.0, 0, 0;
                CMP_SEAS_DUMMY, 1.0, 12, 0;
                CMP_IRREG, 1.0, 0, 0
        >;
//      ranseed(1357924680);
        ranseed(2468013579);
        g_STSiBFGS=50;
        g_STSiBoot=500;
//////////////////////////////////////////////////////////////
//      Local Level Model
//////////////////////////////////////////////////////////////
        STSvYt=loadmat("ipca.dat");
        STSvYt=STSvYt';
        println("Data:\n", STSvYt);
        println("STSAdjust:STSAdjust()");
        STSmOmega=STSAdjust(STSvYt, STSmModelLLM);
        STSmRes=STSBootCI(STSvYt, STSmModelLLM);
        fSigma2EtaEmv=STSmOmega[0][0];
        fSigma2EpsEmv=STSmOmega[rows(STSmOmega)-1][rows(STSmOmega)-1];
        fSigma2EtaBot=meanr(STSmRes[0][:])[0];
        fSigma2EpsBot=meanr(STSmRes[rows(STSmRes)-1][:])[0];
        fSigma2EtaLinf=STSQuantile(STSmRes[0][:],(1-fAlpha)/2);
        fSigma2EtaLsup=STSQuantile(STSmRes[0][:],(1+fAlpha)/2);
        fSigma2EpsLinf=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1-fAlpha)/2);
        fSigma2EpsLsup=STSQuantile(STSmRes[rows(STSmRes)-1][:],(1+fAlpha)/2);
        println("Adjust:\nParameter\tEstimate\tBootEstimate\tLInf\tLSup");
        println("Sigma2Eta\t",
                fSigma2EtaEmv, "\t", fSigma2EtaBot, "\t",
                fSigma2EtaLinf, "\t", fSigma2EtaLsup);
        println("Sigma2Eps\t",
                fSigma2EpsEmv, "\t", fSigma2EpsBot, "\t",
                fSigma2EpsLinf, "\t", fSigma2EpsLsup);
        exit(0);
}
```

Apêndice D

D.1 Série Histórica do IPCA (%) de Belo Horizonte

Período	IPCA (%)
jan/97	1,92
fev/97	0,88
mar/97	-0,06
abr/97	0,69
mai/97	0,55
jun/97	0,17
jul/97	-0,09
ago/97	0,03
set/97	0,19
out/97	0,62
nov/97	0,32
dez/97	0,81
jan/98	0,77
fev/98	0,14
mar/98	-0,04
abr/98	0,28
mai/98	0,63
jun/98	-0,08
jul/98	-0,39
ago/98	-0,59
set/98	0,05
out/98	-0,10
nov/98	0,25
dez/98	0,27
jan/99	0,01
fev/99	1,45
mar/99	0,93
abr/99	0,44
mai/99	0,54
jun/99	0,47
jul/99	1,29
ago/99	0,22
set/99	0,35
out/99	0,59
nov/99	1,38
dez/99	0,94
jan/00	0,63
fev/00	0,04
mar/00	-0,05
abr/00	0,23
mai/00	0,27
jun/00	0,17
jul/00	1,74
ago/00	1,35
set/00	0,13
out/00	0,12
nov/00	0,08
dez/00	1,13
jan/01	0,50
fev/01	-0,01
mar/01	0,62
abr/01	1,38
mai/01	0,28
jun/01	-0,12
jul/01	0,60
ago/01	0,53
set/01	0,19
out/01	0,87
nov/01	0,38
dez/01	0,67
jan/02	0,84
fev/02	0,13
mar/02	0,18
abr/02	0,88
mai/02	0,19
jun/02	0,34
jul/02	0,77
ago/02	0,94
set/02	0,56
out/02	1,56
nov/02	2,83
dez/02	1,51
jan/03	2,52
fev/03	1,07
mar/03	1,41
abr/03	1,00
mai/03	1,23
jun/03	-0,67
jul/03	0,25
ago/03	0,49
set/03	0,45
out/03	0,41
nov/03	-0,07
dez/03	0,46
jan/04	0,98
fev/04	0,31
mar/04	0,48
abr/04	0,49
mai/04	1,25
jun/04	0,59
jul/04	0,50
ago/04	1,05
set/04	0,12
out/04	0,14
nov/04	1,08
dez/04	1,43
jan/05	0,50
fev/05	0,16
mar/05	0,74
abr/05	0,96
mai/05	0,76
jun/05	-0,22
jul/05	0,26
ago/05	-0,12
set/05	0,13
out/05	0,41

REFERÊNCIAS BIBLIOGRÁFICAS

Cramer, J.S. (1986). *Econometric Applications of Maximum Likelihood Methods*. Cambridge: Cambridge University Press.

Davis, R.A. & Yam, G.R. (2003). *Approximate Likelihood Approach.* Colorado: Colorado State University.

Doornik, J.A. (1999). *Ox: An Object-Oriented Matrix Language*. 3. ed. London: Timberlake Consultants Press.

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

Efron, B. (1979). "Bootstrap methods: another look at the jackknife". *The Annals of Statistics*, **7**, 1-26.

Efron, B. & Tibshirani, R. (1986). "Bootstrap methods for standard errors, confidence intervals and other measures of statistica accuracy". *Statistical Csience*, **1**, 54-77.

_____ (1993). *An introduction to the bootstrap.* London: Chapman & Hall.

Fletcher, R. (1987). *Practical Methods of Optimization*, 2nd edition. New York: John Wiley & Sons.

Franco, G.C. (1998). *Bootstrap em Modelos Estruturais: Construção de intervalos de confiança e testes de hipóteses.* Tese (Doutorado em Engenharia Elétrica) – PUC, Rio de Janeiro.

Franco, G.C. & Souza, R.C. (2002). "A Comparison of Methods for Bootstrapping in the Local Level Model". *Journal of Forecasting.* **21**, 27-38.

Gill, P. E., Murray, W. & Wright, M. H. (1981). *Practical Optimization*. New York: Academic Press.

Harvey, A.C. (1989). *Forecasting, structural time series models and the Kalman filter.* Cambridge: University Press.

_____ (2001). "Testing unobserved component models". *Journal of Forecasting*. **20**, 1-19.

Harvey, A.C., Koopman, S.J. & Shephard, N. (2004). *State Space and Unobserved Component Models: Theory and Applications*. Cambridge: University of Cambridge.

Kalman, R.E. (1960). "A new approach to linear filtering and prediction problems". *Journal Basic Engineering, Transactions ASME*. *Series D*, **82**, 35-45.

Olsson, J. (2005). *On Estimation in State Space Models Using the Bootstrap Particle Filter*. Sweden. Lund Institute of Technology: Lund University.

Pfeffermann, D. & Tiller, R. (2004). *Bootstrap Approximation to Prediction MSE for State-Space Models with Estimated Parameters.* (S3RI Methodology Working Papers, M03/05) Southampton, UK, Southampton Statistical Sciences Research Institute.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. (1988). *Numerical Recipes in C*. New York: Cambridge University Press.

Stoffer, D. S. & Wall, K. D. (1991). "Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter". *Journal of American Statistics Association*, **86**, 1024-1033.