Estatística computacional - Trabalho Final

Arthur Sonntag Kuchenbecker

Novembro 2019

1 Introdução

O objetivo do trabalho é aplicar os conhecimentos adquiridos ao longo da disciplina para (i) estimar os parâmetros de uma regressão linear via método MCMC (Metropolis Hastings); e (ii) utilizar o método "moving blocks" (bootstrap) para modelagem de uma série temporal.

Isso será feito por meio do software `R`, conforme explicado nas seções seguintes.

2 Metropolis Hastings para regressão linear

2.1 O processo gerador de dados

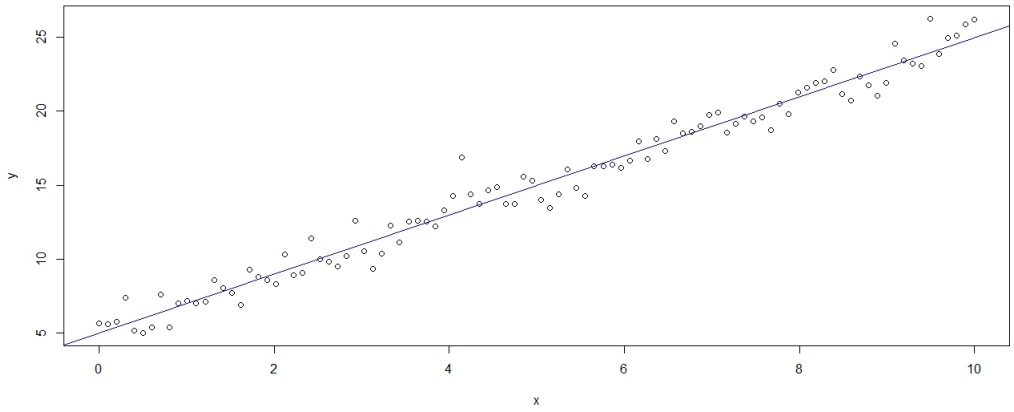
Para gerar os dados foram utilizados os parâmetros $\beta_0 = 5$ e $\beta_1 = 2$. Para o erro, foram gerados valores aleatórios com distribuição normal padrão:

$$\epsilon \sim N(0, 1) \tag{1}$$

O modelo utilizado, portanto, é dado pela equação:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon \tag{2}$$

Abaixo está o gráfico com os dados gerados e a reta que representa a estimativa dos parâmetros da função via MQO (em azul).



2.2 Função verossimilhança

Para estimar parâmetros em uma análise bayesiana, é necessário derivar a função de verossimilhança do modelo de interesse.

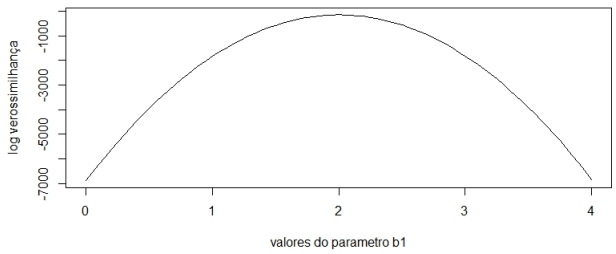
A função de verossimilhança tem como input um valor para os parâmetros do modelo, e retorna a probabilidade de que seja obtida a amostra observada, levando em conta que o modelo possui uma distribuição teórica determinada.

Abaixo segue o código utilizado para criar a função de verossimilhança.

```
vero <- function(p){
       b0_hat = p[1]
       b1_hat = p[2]
       de_hat = p[3]

       y_hat = b0_hat+b1_hat*x

       v = dnorm(y, mean= y_hat, sd= de_hat, log= T)
       soma_v = sum(v)

       return(soma_v)
     }
```

Ao derivar a função de verossimilhança, convêm utilizar o log das probabilidades, o que nos permite retornar a soma das probabilidades de cada observação da amostra. Isso é padrão quando utilizamos funções de verossimilhança, visto que a multiplicação de várias probabilidades deve ir pra zero relativamente rápido, o que dificultaria a análise.

Para ilustrar, abaixo está o gráfico da função de log-verossimilhança para o parâmetro β_1.



2.3 Priori e posteriori

O cerne da inferência bayesiana está em definir uma destribuição teórica para cada um dos parâmetros avaliados. Esta distribuição se chama distribuição a priori e reflete as crenças anteriores à análise dos dados.

Caso não haja nenhuma hipótese inicial para o modelo, recomenda-se utilizar prioris pouco informativas. Neste trabalho utilizou-se a distribuição normal com média igual ao valor do parâmetro e desvio-padrão 1.

```
priori <- function(p){
  b0_hat = p[1]
  b1_hat = p[2]
  de_hat = p[3]

  b0_prior = dnorm(b0_hat, sd=1, log= T)
  b1_prior = dnorm(b1_hat, sd=1, log= T)
  de_prior = dnorm(de_hat, sd=1, log= T)

  return(b0_prior + b1_prior + de_prior)
}
```

A distribuição a posteriori é dada pela fórmula da Bayes:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{3}$$

Na estimação dos parâmetros do modelo linear, estamos interessados em $P(\theta|x)$. É relativamente fácil calcular o numerador da equação (3), como vimos até agora.

Para encontrar $P(\theta|x)$ por inteiro, utilizaremos o algoritmo MCMC. Antes, portanto, ao longo do código de implementação chamaremos de `nposteriori` apenas o numerador da equação (3), dado pela soma da função de log-verossimilhança e o log da probabilidade $P(\theta)$ (priori dos parâmetros).

```
nposteriori <- function(p){

    return(vero(p) + priori(p))
}
```

2.4 A implementação do algoritmo

A implementação do algortimo se dá da seguinte maneira:

1. Valor inicial para os parâmetros;
2. Escolha de um novo valor, perto do valor antigo, baseado em alguma distribuição de probabilidade;
3. Pular para este novo valor caso a divisão da `nposteriori` do novo valor pela `nposteriori` do valor atual seja maior que um valor aleatório entre 0 e 1.

Voltando a equação (3), entendemos o motivo pelo qual este algoritmo funciona para "criar" uma amostra da distribuição posteriori $P(\theta|x)$.

$$\frac{P(\bar{\theta}|x)}{P(\dot{\theta}|x)} = \frac{\frac{P(x|\bar{\theta})P(\bar{\theta})}{P(x)}}{\frac{P(x|\dot{\theta})P(\dot{\theta})}{P(x)}} = \frac{P(x|\bar{\theta})P(\bar{\theta})}{P(x|\dot{\theta})P(\dot{\theta})} \tag{4}$$

Em palavras, dividir a posteriori da novo valor ($\bar{\theta}$) pela posteriori do valor atual do parâmetro ($\dot{\theta}$), faz com que aquela quantidade $P(x)$ - a qual não é obtida facilmente de outro modo - seja cancelada. Sendo assim, podemos concluir que estamos, de fato, dividindo a posteriori em uma posição pela posteriori em outra posição.

Desta forma, o algoritmo chega em regiões da distribuição especificada com alta probabilidade posteriori mais vezes do que em regiões com baixa probabilidade posteriori.

Abaixo segue o código utilizado para implementação do algoritmo.

```
# Desvios padroes das distribuicoes utilizadas para amostrar
    MCMC_de = c(0.1,0.1,0.1)

    # Funcao proposta utilizada para gerar as amostras i+1
    # que serao comparadas com os valores de i

      funcao_proposta <- function(p){

        aux = rnorm(3,mean= p, sd= MCMC_de)
        return(c(aux[1],aux[2],abs(aux[3])))
      }

    # Loop para implementar o algoritmo

    metropolis_MCMC <- function(valor_inicial, it){

      # Cria a matriz da cadeia
      cadeia = array(dim = c(it+1,3))

      # Atribui os valores iniciais
      cadeia[1,] = valor_inicial

      # Loop para gerar as amostras e comparar
      for (i in 1:it){
        proposta = funcao_proposta(cadeia[i,])

        prob = exp(nposteriori(proposta) -
                     nposteriori(cadeia[i,]))

        # Aceita a nova amostra se prob for maior que uma
        # probabilidade aleatoria

        if (runif(1) < prob){

          cadeia[i+1,] = proposta

        }else{

          cadeia[i+1,] = cadeia[i,]
        }
```

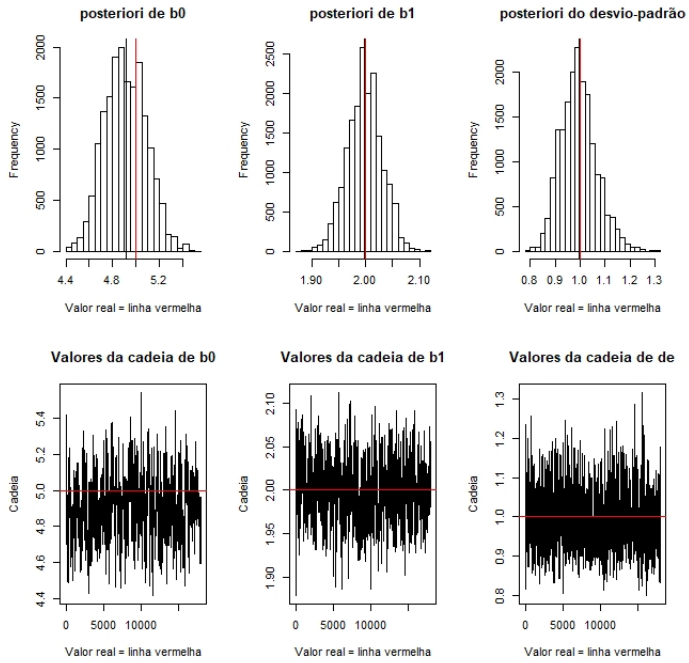
```
    }
    return(cadeia)
}
```

2.5 Estimação

Os valores iniciais para os parâmetros (β_0, β_1 e desvio-padrão) escolhidos foram (1, 1 e 1). O número de iterações para o algoritmo é de 100 mil.

Para diminuir o viés do valor inicial e diminuir a autocorrelação da amostra, utilizamos um período de burn-in de 10 mil observações e um step-size igual a 5.

Abaixo estão os plots com as ditribuições posteriori geradas pelo algoritmo Metropolis Hastings - MCMC.



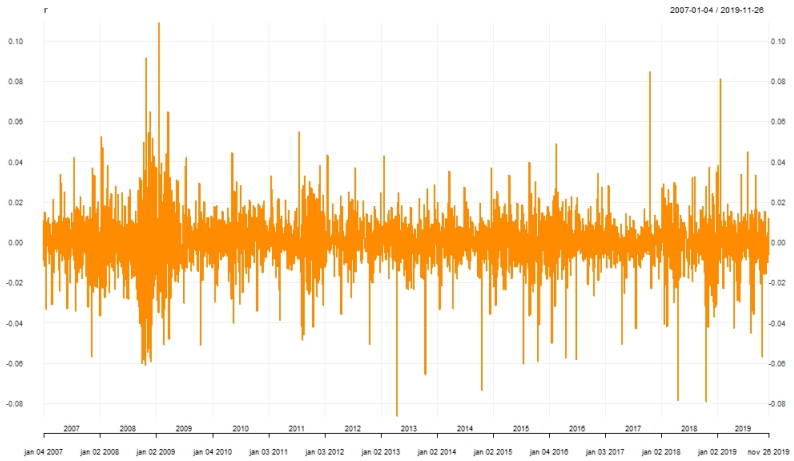
3 Bootstrap para modelagem de séries de tempo

Nesta seção utilizaremos o método "moving blocks" para modelar uma série de tempo financeira com bootstrap.

O bootstrap, em uma breve explicação, é um método computacionalmente intensivo utilizado para reamostrar várias vezes de uma mesma amostra, gerando assim uma distribuição para determinados parâmetros que se busca estimar. Este método é bastante utilizado quando não é possível obter amostras grandes o suficiente para fazer a inferência.

No presente trabalho, este não é o caso, mas o método bootsrap com "moving blocks" será implementado para comparar o resultado com aquele obtido através da estimação direta baseada da amostra.

A série utilizada é a série histórica do preço das ações da IBM, desde janeiro de 2007 até hoje. Segue abaixo gráfico da série de preços e da série de retornos.



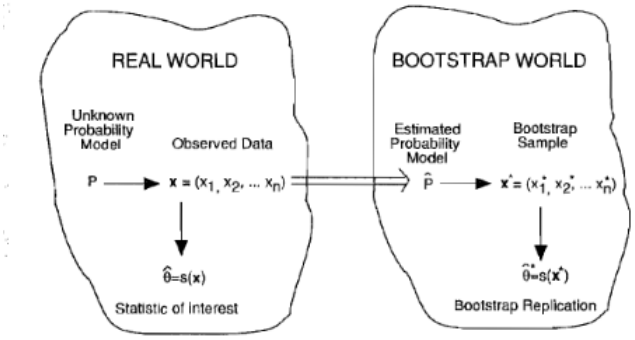
3.1 Implementação

O código feito é baseado na na seção 8.6 do livro de Bradley Efron e R.J. Tibshirani (1994)

Um resumo da teoria utilizada: enquanto o bootstrap para amostras que não são séries temporais funciona se obtivermos várias reamostragens apenas alterando a ordem de cada observação da amostra original, para séries de tempo essa reamostragem elemento a elemento faz com que a estrutura de autocorrelação da série original se perca.

Dessa forma, para bootstrap de séries temporais, o método "moving blocks" é interessante pois faz a reamostragem por blocos (grupos ordenados) de observações.

As imagens abaixo foram retiradas do livro citado e são bastante elucidativas para entender a aplicação aqui realizada.



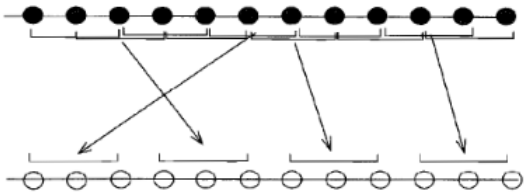


Figure 8.9. *A schematic diagram of the moving blocks bootstrap for time series. The black circles are the original time series. A bootstrap realization of the time series (white circles) is generated by choosing a block length ("3" in the diagram) and sampling with replacement from all possible contiguous blocks of this length.*

3.2 Código e resultados

Na prática, a implementação funcionou da seguinte forma:

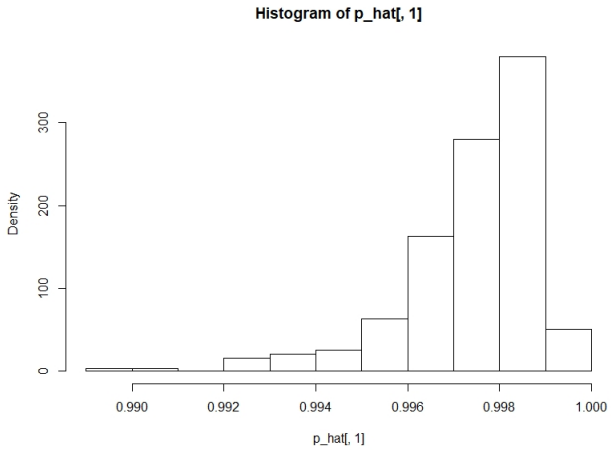
1. Busca da série temporal (preço das ações IBM);
2. Tratamento da série de preços (transformação em uma série de retornos, estacionária);
3. Reamostragem da série de retornos, mas a cada iteração do algoritmo se utiliza a série de retornos para obter uma nova série de preços (partindo do mesmo preço inicial);
4. Comparação do parâmetro AR(1) estimado para a série original de preços com a média da cadeia de parâmetros gerados a partir do algoritmo bootstrap.

Abaixo está a parte do código que faz a implementação do algoritmo.

```
# Estima modelo AR(p) para as series
# (y - precos; r - retornos)
    ar(y)
    ar(r)

    # Tamanho da serie dos retornos original
    n = length(r)

    # Tamanho dos "moving blocks"
    k = 40

    # Numero de repeticoes (cada repeticao e uma
    # reamostragem da amostra original)
    nr = 400

    # Cria matriz para salvar
```

```
# os coeficientes AR(p) estimados
p_hat = array(dim = c(nr,8))

# Loop para a reamostragem em blocos

for(i in 1:nr){

  # Cria o vetor para as series reamostradas
  r_bt = rep(NA,n)

  # Loop especifico para lidar com os blocos
  for(j in 1:ceiling(n/k)){

    fim = sample(k:n, size=1)
    r_bt[(j-1)*k+(1:k)] = r[fim-(k:1)+1]
  }

  # Trunca para o caso em que k nao divide n
  r_bt = r_bt[1:n]

  # Estima os coeficientes do AR(p)

    # Recria a serie y_bt (precos),
    # com base em r_bt (retornos)
    y_bt = y

    for (q in 2:n+1){
      y_bt[q] = exp(log(y_bt[q-1])+r_bt[q-1])
    }

    # Trunca a ordem do AR(p) para no maximo 8
    ordem_max = ar(y_bt)$order

    if (ordem_max > 8){
      ordem_max = 8
    }

    for (l in 1:ordem_max){

      p_hat[i,l] <- ar(y_bt)$p[l]
    }
}
```

O histograma da cadeia de parâmetros AR(1) estimados está abaixo:



Por fim, a comparação entre a média dos parâmetros estimados para cada uma das reamostragens bootstrap com o parâmetro estimado da série original:

```
# Compara a media das estimativas bootstrap
# com a estimativa da serie original

>       mean(p_hat[,1])
[1] 0.9974519

>       ar(y)$p[1]
[1] 0.9976683
```