

Brazilian Economic Time Series (BETS): R package

Pedro Costa Ferreira, Talitha F. Speranza, Jonatha A. da Costa,

1. Introdução

O pacote BETS (sigla para Brazilian Economic Time Series) para o R ([R Core Team, 2012](#)) fornece, de maneira descomplicada, as mais relevantes séries temporais econômicas do Brasil e diversas ferramentas para analisá-las. O BETS preenche uma lacuna no processo de obtenção de dados no Brasil, na medida em que unifica os pontos de acesso às séries e oferece uma interface bastante simples, flexível e robusta.

As séries presentes na base de dados do pacote são produzidas por três importantes e respeitados centros: o Banco central do Brasil (BACEN), o Instituto Brasileiro de Geografia e Estatística (IBGE) e o Instituto Brasileiro de Economia da Fundação Getúlio Vargas (FGV/IBRE). Em sua concepção original, o BETS pretendia reunir em um só lugar o maior número possível de séries destes centros, dada a dificuldade com que o pesquisador poderia se defrontar para obtê-las. Este objetivo foi cumprido e hoje o pacote disponibiliza mais de 39 mil séries econômicas brasileiras.

Por conta do vasto tamanho da base de dados, o BETS foi expandido para prover mecanismos que auxiliem o analista na pesquisa, extração e exportação das séries. A partir daí a formulação do pacote se transformou e o BETS caminhou para tornar-se um ambiente integrado de análise e aprendizado. Foram incorporadas diversas funcionalidades ausentes do universo R, com a intenção de facilitar ainda mais a modelagem e a interpretação das séries fornecidas. Ademais, algumas funções já incluem a opção de gerar saídas inteiramente didáticas, incentivando e auxiliando o usuário a ampliar seus conhecimentos.

Ao longo do capítulo será descrito a estrutura e mostrado algumas das principais funcionalidades do BETS. Alguns exemplos típicos de uso também serão exibidos, passo a passo. Em seguida, o arcabouço do pacote será visto em mais detalhe, explicando como ele está dividido atualmente. Introduzimos, também, as formas básicas de uso do BETS, mostrando como é feita a busca, a leitura e o armazenamento das séries. As funções mais avançadas serão apresentadas mais adiante, junto com dois estudos de caso completos: a análise da série de produção de bens intermediários do Brasil e do Índice de Preços ao Consumidor Amplo (IPCA), utilizando as ferramentas do BETS.

2. Banco de dados e estrutura do pacote

2.1 Banco de dados

A figura 1 esquematiza o banco de dados do BETS tal como é hoje. Cerca de 61% da base é composta pelas séries produzidas pelo FGV/IBRE, ao passo que as 39% restantes pertencem ao IBGE e ao BACEN.

A maior parte dos dados foi coletada através de técnicas de web scraping, baseadas no trabalho de Rademaker (2012). Utilizando os pacotes SSOAP (Lang, 2012a) e XMLSchema (Lang, 2012b) para o R, implementamos uma extensa API para extrair os dados automaticamente da página do BACEN a partir de scripts em R. As séries do IBGE disponíveis no BETS também foram obtidas na página do BACEN. Um resumo esquemático da abordagem que seguimos pode ser encontrado na figura 2.

Para as séries do FGV/IBRE, esse procedimento não foi necessário, pois os dados foram fornecidos diretamente pela instituição, que apoia o projeto.

Tradicionalmente, o analista poderia obter as séries do IBRE no sistema FGV Dados, onde estão presentes séries livres e pagas, além de toda a produção estatística da FGV e um conjunto selecionado de indicadores de outros centros. Para adquirir as séries pagas, é necessário ser assinante e não apenas cadastrado. Com o BETS, é possível acessar todas estas séries, porém com uma defasagem de vinte meses.

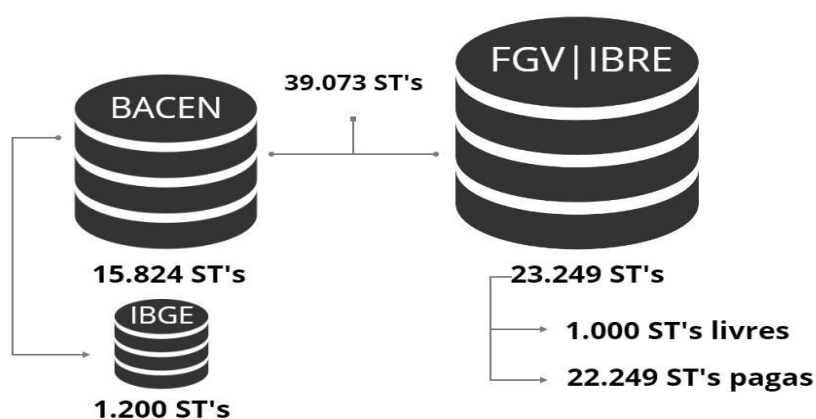


Figura 1: Séries presentes no banco de dados do BETS

Uma discussão importante durante a fase de concepção do BETS foi a localização da base de dados. Da forma que é hoje, a base é importada juntamente com o pacote, em arquivos .rda. Portanto, é necessário atualizar o pacote para receber os novos dados. Esta arquitetura dispensa o uso de um

servidor, o que é vantajoso do ponto de vista da implementação e da manutenção. Por outro lado, ela é menos prática para o usuário. Caso tivéssemos construído um banco de dados com algum sistema gerenciador (SGBD) em um servidor, a atualização dos dados seria feita por uma equipe de mantenedores diretamente neste servidor, livrando o usuário da obrigação de obter os novos dados manualmente. Por este motivo, a base não será distribuída juntamente com o BETS em versões posteriores.

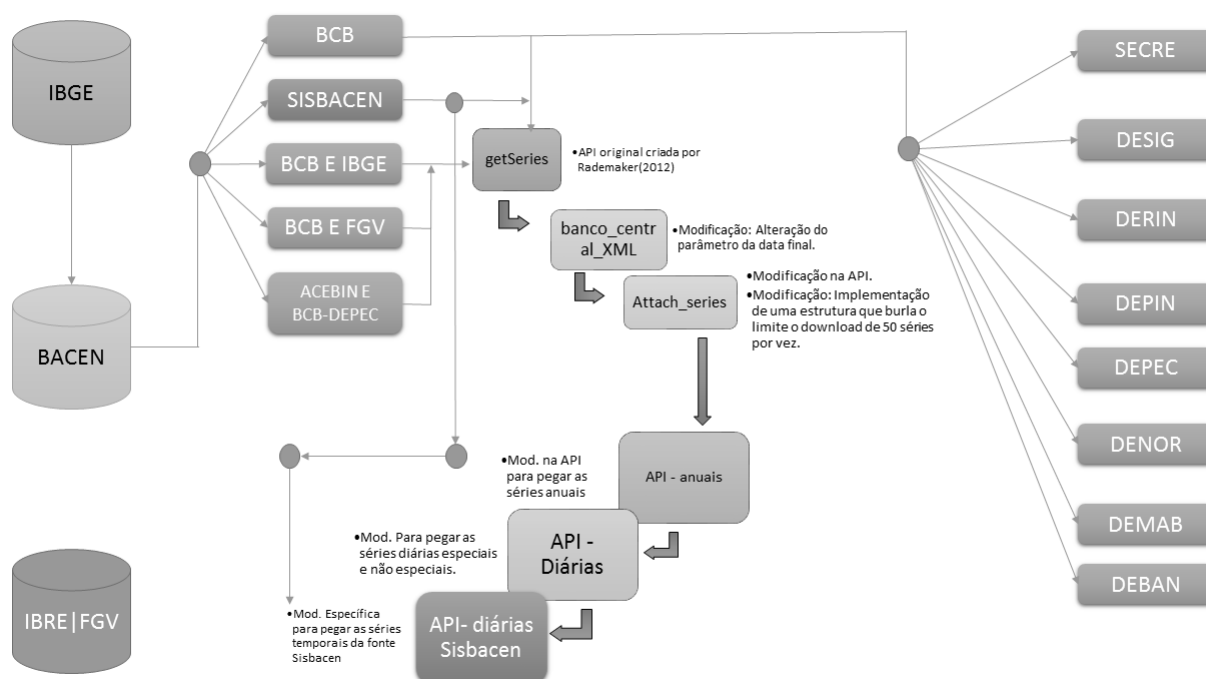


Figura 2: Organograma do procedimento adotado para obter as séries presentes no BETS.

Como já foi ressaltado, uma das principais vantagens do BETS é a facilidade de acesso às mais de 39 mil séries históricas em sua base. Isto não teria sido possível sem uma tabela de metadados que possuísse informações como a descrição, a periodicidade, a unidade em que os dados foram representados e as datas de início e fim das séries. Há, além desta importante tabela, outras cinco. Elas contêm, respectivamente, séries mensais, anuais, semanais, diárias e trimestrais, como mostra o modelo conceitual da figura 3.

Tal divisão foi feita para agilizar internamente a interação do usuário com o banco de dados. Com o mesmo objetivo, demos a cada série um código único, tratado como índice pelo pacote auxiliar sqldf (Grothendieck, 2015). Este pacote manipula os *data frames* do R com SQL, o que significa que realiza buscas por colunas e possui alto desempenho em termos de tempo de processamento.

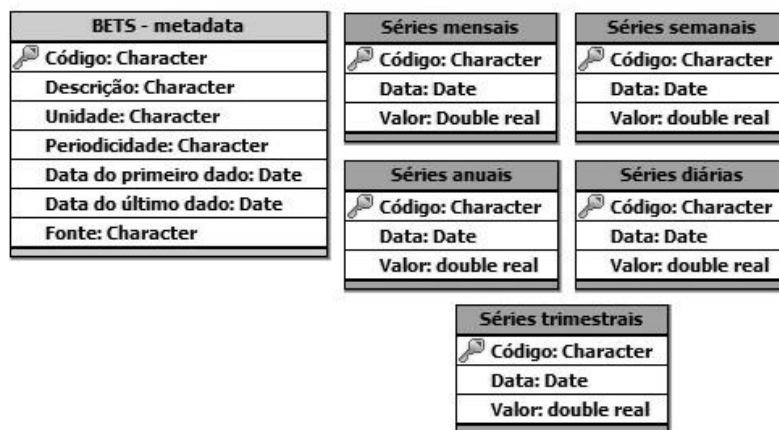


Figura 3: Modelo conceitual da base de dados do BETS

2.2 Estrutura do Pacote

A estrutura do BETS é parecida com a de uma DMP (Data Management Platform). O conceito de DMP é mais aplicado aos grandes sistemas de coleta, armazenamento e processamento de dados direcionados ao marketing, mas guarda fortes semelhanças com o que é apresentado aqui. O DMP pode ser definido simplesmente como uma estrutura onde os dados são centralizados, mesclados e acionados por diversas plataformas. A composição desta estrutura se dá em três etapas. Primeiro, os dados são coletados por uma API externa. Então, eles são organizados de acordo com determinados temas e de modo a otimizar o tempo de acesso dos algoritmos de busca. Finalmente, o acionamento das informações pelo usuário final se dá através de alguma outra API.

Anteriormente cobrimos as etapas de obtenção e organização dos dados. Cabe, agora, mostrar de que maneira o pacote foi estruturado, para então explicar como o usuário final acessa as bases.

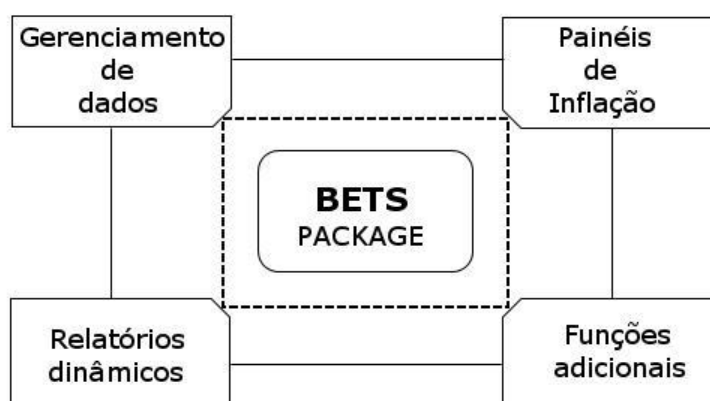


Figura 4: Grupos de funcionalidades do BETS

As funcionalidades do BETS podem ser divididas em quatro grupos (figura 4). São eles:

- **Gerenciamento de dados:** ferramentas para obter as séries e informações sobre elas, englobando tanto a API privada de extração dos dados diretamente das fontes quanto a API pública de recuperação dos dados da base do pacote.
- **Relatórios dinâmicos:** documentos que reportam a análise e as previsões de uma série à escolha, de acordo com algum método bem estabelecido. São gerados automaticamente, bastando que o usuário forneça o código da série nas bases do BETS e alguns parâmetros adicionais. Atualmente, estão disponíveis a análise de modelos SARIMA e a aplicação de redes neurais do tipo GRNN (ver seção 2.5). Os documentos possuem, obrigatoriamente, comentários didáticos. No entanto, a próxima versão do BETS dará a opção de gerar relatórios puramente técnicos.
- **Painéis de Inflação:** documento de análise de conjuntura, contendo uma seleção de gráficos estilizados das séries históricas mais utilizadas para prever e controlar a inflação no Brasil. Em versões futuras, o BETS incluirá painéis para outros países e abrangendo outros temas além da inflação.
- **Funções adicionais:** completam o escopo do pacote, introduzindo métodos que auxiliem os analistas e, de modo mais geral, ajudem o usuário a interagir com as informações contidas nas séries.

O BETS possui, ainda, um outro aspecto funcional que não se encaixa nos grupos apresentados, por atuar uma única vez. Quando o pacote é instalado, diversos pacotes úteis à análise de dados também são instalados. Uma lista completa destes pacotes é exibida na tabela 1. A ideia é que o ambiente do usuário fique totalmente equipado para a manipulação das séries temporais disponíveis. Vários destes pacotes são utilizados para implementar o próprio BETS, como será visto ao longo do texto.

Nome	Descrição	Autor
urca	Implementação dos testes de raiz unitária e cointegração na análise econométrica aplicada.	Pfaff et al. (2016)
forecast	Métodos e ferramentas para exibição e análise univariada de previsões de séries temporais, incluindo suavização exponencial através de modelos de espaço de estado e modelagem automática de modelo ARIMA.	Hyndman (2015)
TSA	Contém funções R e conjuntos de dados detalhados no livro "Time Series Analysis with Applications in R (second edition)" por Jonathan Cryer e Chan Kung-Sik.	Chan and Ripley (2012)
ggplot2	ggplot2 é um sistema de plotagem de R, com base na gramática de gráficos.	Wickham and Chang (2015)
plotly	Traduz gráficos feitos com o pacote 'ggplot2' para uma versão interativa baseada na web e/ou cria visualizações personalizadas baseadas na web diretamente a partir de R.	Sievert et al.(2016)
TTR	Funções e dados para construir as regras técnicas de negociação com R.	Ulrich (2016)
FinTS	Análise de séries temporais financeiras.	Graves (2014)
stringi	Disponibiliza: padrão de pesquisa (por exemplo, Java de ICU, como expressões regulares ou o algoritmo de agrupamento Unicode), geração de sequência aleatória, mapeamento de processos, transliteração de sequência de caracteres, concatenação, normalização Unicode, formatação de data e hora, etc.	Gagolewski and Tartanus (2016)
sqldf	Manipula base de dados no R usando SQL.	Grothendieck (2015)
foreign	Funções para ler e gravar dados.	Bivand et al.(2016)
lmtest	Uma coleção de conjuntos de dados, testes e exemplos para verificação de diagnóstico em modelos de regressão linear.	Hothorn et al.(2016)
normtest	Testes para a hipótese composta de normalidade.	Gavrilov and Pusev (2016)
zoo	Uma classe S3 com métodos totalmente ordenados para observações indexadas. Destina-se particularmente a séries irregulares de vetores/matrizes numéricas e fatores. As metas de design chave do pacote são a independência de uma determinada classe de índice/data/hora e consistência com ts e base R, fornecendo métodos para estender padrões genéricos.	Zeileis et al. (2016)
rugarch	Modelos GARCH univariados.	Ghalanos (2016)
xlsx	Fornecer funções em R para ler/escrever em formato Excel 2007 e formatos de arquivo do Excel 97/2000/XP/2003.	Dragulescu (2016)

Tabela 1: Alguns dos pacotes que são instalados com o BETS

A interface do usuário com o banco de dados é bastante intuitiva. Há uma função para busca, uma para extração de dados e uma classe de funções para armazenamento externo (tabela 2), para que os dados sejam processados por softwares populares como o Microsoft Excel, o SaS, o Stata ou o SPSS.

Nome	Descrição
BETS.search	Realiza pesquisa de séries por diferentes características.
BETS.get	Extraí totalmente a série do banco de dados e carrega para o ambiente R
BETS.save.spss	Exporta a série temporal em um arquivo com extensão .spss
BETS.save.sas	Exporta a série temporal em um arquivo com extensão .sas
BETS.save.stata	Exporta a série temporal em um arquivo com extensão .dta
BETS.save.xlsx	Exporta a série temporal em um arquivo com extensão .xlsx

Tabela 2: Funções para a interação do usuário com o banco de dados do BETS

Com as informações apresentadas até aqui, fica certamente mais nítida a promessa que o BETS representa. Extrapolando o básico fornecimento de dados com uma quantidade considerável de ferramentas para o estudo das séries, o pacote é pioneiro em permitir que qualquer programador iniciante, de qualquer parte do mundo, tenha condições de analisar o cenário econômico do país. Todo o poder do BETS estará ao passo de um simples comando de instalação, o `install_github`.

```
> indevtools::install_github("pedrocostaferreira/BETS")
```

É importante informar que pode aparecer certos problemas na hora da instalação pelo Github. Nesse repositório o BETS não possui credencial para manipular e instalar todas as dependências dos pacotes que são necessários para o funcionamento completo do BETS. Logo, é preciso instalá-los manualmente. Acesse o repositório no GitHub (pedrocostaferreira/BETS) pois lá contém os códigos necessários para suprir essa necessidade. Tal problema será corrigido assim que o BETS for submetido e aceito no CRAN.

3. Utilizando o BETS

Discutiremos algumas formas básicas de uso do pacote. As funções que não são apresentadas nesta seção serão tratadas nos estudos de caso, com aplicações reais. Já as funções possivelmente mais importantes do pacote figurarão em ambas as seções, porém o nível de detalhamento será maior aqui.

Em primeiro lugar, é necessário instalar e carregar o BETS. Atualmente, o pacote só está disponível no GitHub, um serviço *web* de hospedagem de repositórios Git, um sistema distribuído de controle de versão. Todos os repositórios públicos do sistema podem ser encontrados em <https://github.com/>. Isto significa que é necessário instalar o pacote devtools previamente, a fim de chamar a função `install_github`.

```
> # Instalar e carregar o devtools
> install.packages("devtools")
> library(devtools)
>
> # Instalar e carregar o BETS
> install_github("pedrocostaferreira/BETS")
> library(BETS)
```

3.1 Gerenciamento de dados

BETS.search

Devido ao tamanho considerável da base de dados, foi necessário criar um modo de pesquisa por séries a partir de seus metadados, isto é, uma ferramenta de busca que utilizasse uma ou mais informações das séries como palavras-chave.

A função `BETS.search` realiza as pesquisas por cada campo da tabela de metadados descrita mais a frente. A `BETS.search` naturalmente possibilita combinações destas características, tornando a busca mais flexível. Ressaltamos que o acesso à base do BETS é feito através do pacote `sqldf`, o que torna o tempo de processamento das buscas suficientemente rápido e mantém a alta performance do pacote em qualquer ambiente.

O protótipo da `BETS.search` tem a forma:

```
BETS.search(description,src,periodicity,unit,code,view = TRUE,lang = "en")
```

Onde os argumentos recebem, respectivamente:

- `description` - Um character. String de busca com os termos que devem ou não estar presentes na descrição da série desejada.
- `src` - Um character. A fonte dos dados.
- `periodicity` - Um character. A frequência na qual a série é observada.
- `unit` - Um character. A unidade na qual os dados foram medidos.
- `code` - Um integer. O código único da série na base do BETS.

- `view` - Um boolean. Por padrão, `TRUE`. Se `FALSE`, os resultados serão mostrados direto no console do R.
- `lang` - Um character. Idioma da pesquisa. Por padrão, `'en'`, para inglês. Também é possível fazer a pesquisa em português, bastando mudar o valor para `'pt'`.

Para refinar as buscas, há regras de sintaxe para o parâmetro `description`:

1. Para procurar palavras alternativas, separe-as por espaços em branco. Exemplo: `description = 'núcleo ipca'` significa que a descrição da série deve conter 'ipca' e 'núcleo'.
2. Para procurar expressões inteiras, basta cercá-las com **aspas simples**. Exemplo: `description = 'índice 'núcleo ipca''` significa que deve conter na descrição da série 'núcleo ipca' e 'índice'.
3. Para excluir palavras da busca, insira um `~` (o caracter til) antes de cada um delas. Exemplo: `description = 'ipca ~ núcleo'` significa que a descrição da série deve conter 'ipca' e não pode conter 'núcleo'.
4. Para excluir todas as expressões da busca, de forma semelhante ao item anterior, basta cercá-los com `~` e inserir um antes de cada uma delas. Exemplo: `description = 'índice ~ 'núcleo ipca''` significa que a descrição da série deve conter 'índice' e **não** pode conter 'núcleo ipca'.
5. É possível pesquisar ou negar várias palavras ou expressões, desde que sejam respeitadas as regras precedentes.
6. O espaço em branco após o sinal de negação (`~`) não é necessário. Mas os espaços em branco depois de expressões ou palavras são necessários.

Alguns exemplos de uso podem ser vistos abaixo. Não é necessário mostrar os resultados em alguns casos, pois a saída pode ser uma tabela demasiada extensa. Contudo, garantimos que todas as chamadas funcionam e convidamos o leitor a testá-las.

```
> BETS.search(description = "sales ~ retail")
> BETS.search(description = "'sales volume index' ~ vehicles")
> BETS.search(description = "'distrito federal'", periodicity = 'A', src = 'IBGE')
> BETS.search(description = "gdp accumulated", unit = "US", view = F)
```

##	Codes	Description
## 1	4192	GDP accumulated in the last 12 months - in US\$ million
## 2	4386	GDP accumulated in the year - in US\$ million
##	Periodicity	start source unit

```
## 1          M 01/31/1990 BCB-Depec US$ (million)
## 2          M 01/31/1990 BCB-Depec US$ (million)

> BETS.search(description = "consumption ~ 'seasonally adjusted' ~ private", view = F)

##      Codes                                     Description
## 1  7332   Quarterly GDP - observed data (1995=100) - Government consumption
## 2  22101  Quarterly GDP - observed data (2010=100) - Government consumption
##      Periodicity      start source      unit
## 1              Q   03/31/1991   IBGE   Index
## 2              Q   03/31/1995   IBGE   Index
```

Para mais informações sobre a `BETS.search`, incluindo os valores válidos em cada campo, consulte o manual de referência, digitando `?BETS.search` no console do R.

BETS.get

A `BETS.get` funciona unicamente através do código de referência da série, obtido com as consultas feita com a `BETS.search`. Sua assinatura é:

```
BETS.get(code, data.frame = FALSE)
```

O parâmetro `code` é, obviamente, obrigatório. O argumento opcional `data.frame` representa o tipo do objeto que será retornado. Por padrão, seu valor é `FALSE`, indicando que o objeto devolvido pela função será um objeto do tipo `ts` (*time series*). Caso `data.frame = TRUE`, a série será armazenada em um objeto do tipo `data.frame`.

Vamos extrair duas das séries pesquisadas anteriormente.

```
> # Obter a serie do PIB acumulado em 12 meses, em dolares
> gdp_accum = BETS.get(4192)
> window(gdp_accum, start = c(2014,1))

##           Jan           Feb           Mar           Apr           May           Jun           Jul           Aug
## 2014 2462906 2468910 2467247 2460576 2458893 2452748 2447154 2437977
## 2015 2365163 2311430 2263498 2210154 2154889 2106212 2051067 1996712
## 2016 1749556 1730334
##           Sep           Oct           Nov           Dec
## 2014 2434635 2428324 2421043 2415916
## 2015 1939427 1880209 1823847 1768770
## 2016

> # Obter a serie do PIB do Distrito Federal, a precos de mercado
> gdp_df = BETS.get(23992, data.frame = T)
> head(gdp_df)

##      dates      GDPamp (2-DF)
## 1 2010-04-07 1.441684e+11
## 2 2011-04-07 1.544683e+11
## 3 2012-04-07 1.638808e+11
## 4 2013-04-07 1.753628e+11
```

BETS.save

Para conferir versatilidade às formas de armazenamento das séries do BETS, há a possibilidade de criar arquivos com as séries em formatos proprietários, isto é, formatos que pertencem a softwares pagos.

Basicamente, a BETS.save extrai a série temporal da base de dados do pacote na forma de um `data.frame` e cria um arquivo no formato especificado. No arquivo, há uma tabela onde a primeira coluna conterá as datas e a segunda, os dados.

A função possui quatro variações:

```
BETS.save.sas(code,data = NULL,file.name = "series")
BETS.save.spss(code,data = NULL,file.name = "series")
BETS.save.xlsx(code,data = NULL,file.name = "series")
BETS.save.stata(code,data = NULL,file.name = "series")
```

Novamente, o parâmetro `code` recebe o código da série. O usuário pode fornecer sua própria série através do argumento `data`, que pode ser um *data.frame* ou um *ts*. Não é necessário acrescentar a extensão ao nome do arquivo no parâmetro `file.name`.

Alguns exemplos típicos de uso seriam:

```
> # Salvar a série da dívida pública líquida no formato padrão do Excel
> BETS.save.xlsx(code = 2078, file.name = "series_excel.xlsx")
>
> # Salvar uma série qualquer no formato do SPSS
> BETS.save.spss(data = myseries, file.name = "series_spss")
```

3.1.1 Algumas Funções Adicionais

A maior parte das funções adicionais será coberta nos estudos de caso. Aqui, falaremos das restantes. Algumas das funções mais especiais do BETS estão entre elas.

BETS.chart

A BETS.chart foi inicialmente projetada para ser uma função privada, auxiliar da BETS.dashboard. No entanto, pode ser de grande valia para o usuário dispor de um meio de obter os gráficos dos dashboards separadamente, de modo a poder incorporá-los em seus trabalhos.

O protótipo da BETS.chart é o que se segue:

```
BETS.chart(alias, lang = "en", out = "png", file = NULL, start =  
           c(2006,1), ylim = NULL)
```

O parâmetro `alias` pode receber uma dentre as várias opções de gráfico. O segundo parâmetro, `lang` é, por padrão, 'en', para inglês. Mas, como no caso da BETS.search, esse valor pode ser modificado para 'pt'. Há também a opção de alterar a extensão da saída para 'pdf', redefinindo o argumento `out`. O parâmetro `start` especifica qual deve ser a data de início da série. Uma vez que se trata de um gráfico de conjuntura, a data de fim não pode ser alterada e é sempre o último dado disponível. Caso o usuário deseje redimensionar o eixo y, pode determinar o valor de `ylim`.

Abaixo dois exemplos de uso da BETS.chart.

```
> BETS.chart(alias = 'ipca_with_core', file = "images/ipca", out = "pdf")
```

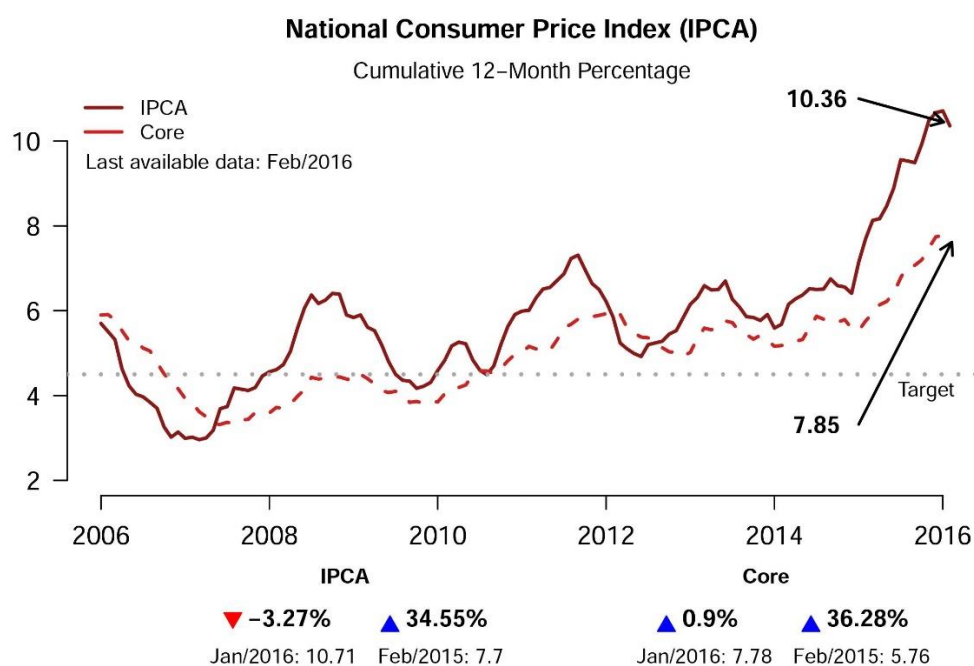


Figura 5: Gráfico do IPCA (Índice Nacional de Preços ao Consumidor Amplo) feito com a BETS.chart.

```
> BETS.chart(alias = 'ulc', start = c(2008,1), file = "images/ulc", out = "pdf")
```

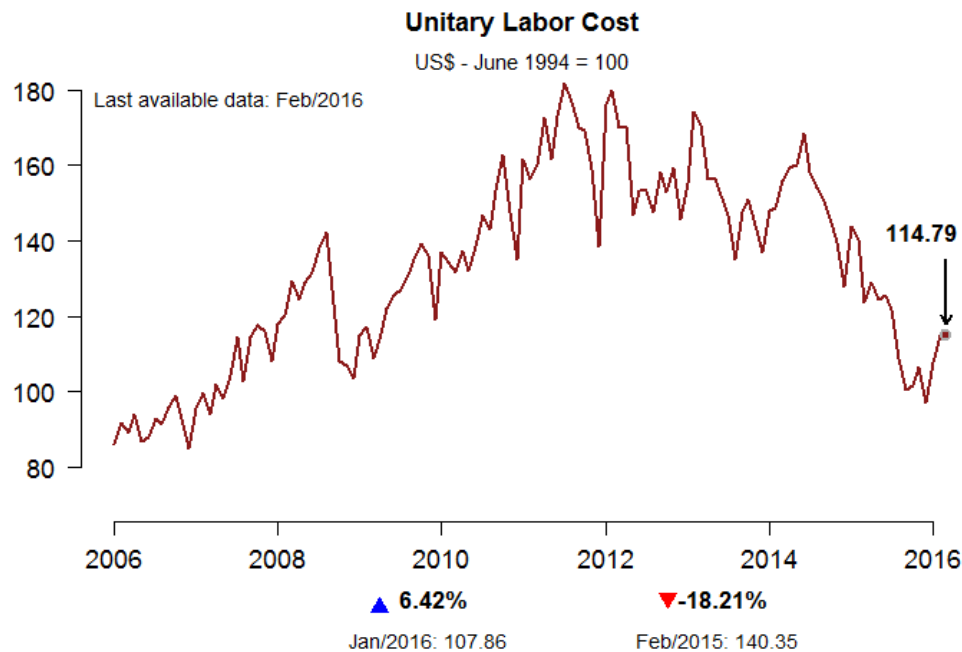


Figura 6: Gráfico do Custo Unitário do Trabalho feito com a BETS.chart

Para uma lista completa dos gráficos disponíveis, consulte o manual de referência da BETS.chart.

3.2 Painéis de inflação

BETS.dashboard

Anteriormente, foi dito que o BETS possui uma poderosa ferramenta de análise de conjuntura, os painéis de inflação. Atualmente, apenas os dados do Brasil estão disponíveis, então só é possível gerar painéis da situação doméstica. Posteriormente, serão incorporados os dados do Banco Mundial e do FMI (Fundo Monetário Internacional), de forma que painéis internacionais poderão ser criados. Também planeja-se expandir o escopo dos painéis para além do tema da inflação.

Vemos abaixo que a estrutura da função já está preparada para receber as versões futuras do BETS. Então, embora os parâmetros `type` e `country` estejam presentes, eles ainda não podem ser modificados.

```
BETS.dashboard(type = "inflation", country = "BR", parameters = NULL, saveas
               = NA)
```

Supondo que tenhamos um arquivo `text.txt` com o texto que desejamos incluir no dashboard, chamamos a `BETS.dashboard` e obtemos um arquivo pdf cuja primeira página será semelhante à

mostrada na figura 7. O texto é opcional. Os gráficos que serão exibidos também são de escolha do usuário, através do parâmetro `charts` (por *default*, seu valor é 'all'). O manual de referência possui uma lista completa dos gráficos disponíveis.

O autor do texto deve fornecer seu nome e seu site e/ou e-mail. O logo não é obrigatório.

```
> parameters = vector(mode = "list")
> parameters$author = "FGV/IBRE"
> parameters$url = "http://portalibre.fgv.br/"
> parameters$logo = "logo_ibre.png"
> parameters$text = "text.txt"
> BETS.dashboard(type = "inflation", parameters = parameters)
```

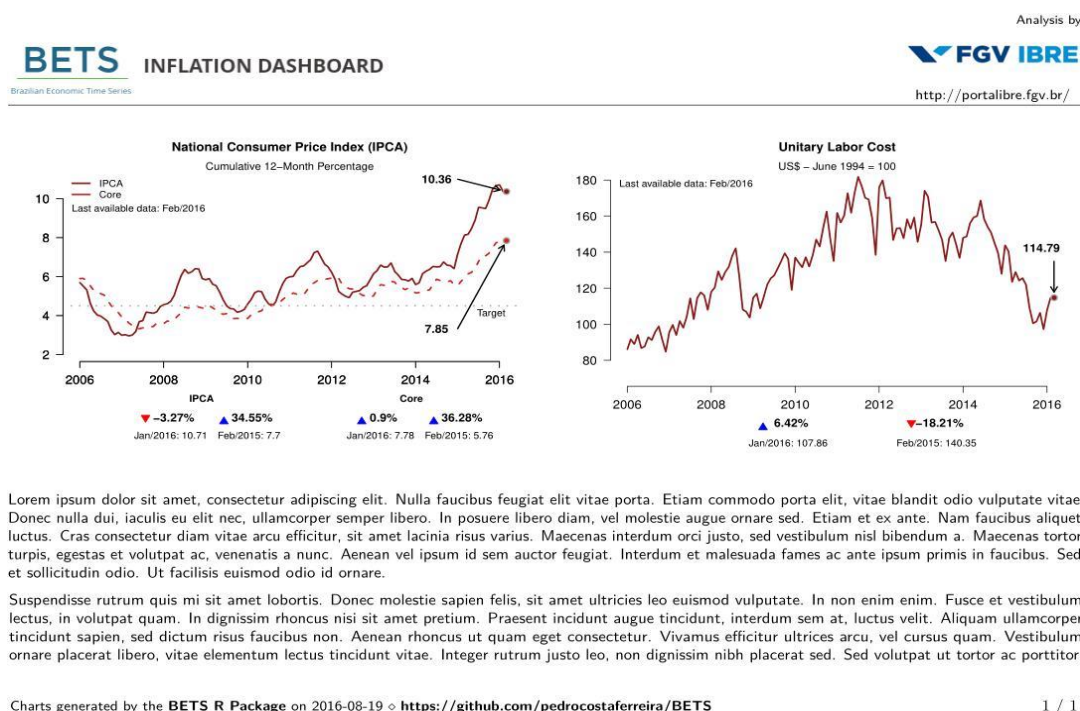


Figura 7: Primeira página do painel de inflação, com texto fornecido externamente através de um arquivo .txt

3.3 Relatórios dinâmicos: modelos Box & Jenkins e Grnn (redes neurais)

A partir de agora serão relatados dois casos típicos de uso do BETS, procurando modelar as séries presentes no pacote a partir da abordagem de Box & Jenkins e de redes neurais. Há uma função no BETS que efetua as análises exatamente da maneira que será feita aqui, com um código idêntico em

muitos pontos. A saída da função é um relatório automático com os resultados e comentários didáticos. Ao longo do texto, será explicado como ela pode ser utilizada.

Antes de começar cada análise, é necessário examinar muito brevemente as metodologias empregadas. Para um tratamento mais abrangente dos tópicos desenvolvidos, consultar [Montgomery et al. \(2015\)](#), [Box and Jenkins \(1970\)](#), [Specht \(1991\)](#) e [Ferreira \(2016\)](#).

3.3.1 Metodologia Box & Jenkins

O método de Box & Jenkins permite que os valores futuros da série em estudo sejam previstos somente com base nos valores passados e presentes da mesma série, isto é, as previsões são feitas a partir de modelos univariados.

Estes modelos são chamados SARIMA, uma sigla para o termo em inglês Seasonal Auto-Regressive Integrated Moving Average, e têm a forma:

$$\Phi_P(B) \varphi_P(B) \nabla^d \nabla^D Z_t = \Theta_Q(B) \theta_Q(B) a_t \quad (1)$$

onde

- Z_t é a série em estudo
- a_t é um ruído branco
- $\nabla^d = (1 - B)^d$ é o operador de diferenças e d o número de raízes unitárias
- $\nabla^D = (1 - B^s)^D$ é o operador de diferenças na frequência sazonal s e D o número de raízes unitárias sazonais
- $\varphi_P(B)$ é o polinômio autorregressivo
- $\Phi_P(B)$ é o polinômio autorregressivo sazonal
- $\theta_Q(B)$ é o polinômio de médias móveis
- $\Theta_Q(B)$ é o polinômio de médias móveis sazonal

Em sua concepção original, que será adotada aqui, a metodologia de Box & Jenkins se divide em três estágios iterativos:

- (i) Identificação e seleção dos modelos: verificação da estacionariedade e da sazonalidade, com as devidas correções para os casos não estacionários e sazonais, e determinação das ordens

dos polinômios descritos acima, fazendo uso das funções de autocorrelação (FAC) e autocorrelação parcial (FACP).

(ii) Estimação dos parâmetros do modelo, usando máxima verossimilhança ou regressão dinâmica.

(iii) Diagnóstico da conformidade do modelo, através de testes estatísticos.

Se o modelo não for aprovado na fase (iii), volta-se ao passo (i). Caso contrário, o modelo pode ser utilizado para fazer previsões. Na próxima seção, conforme o exemplo for evoluindo, cada um desses estágios será observado de perto e mais será dito sobre a metodologia.

Preliminares

Vamos trabalhar com a série de código 21864, a série de produção de bens intermediários do Brasil (PBI). O primeiro passo é encontrá-la na base de dados do BETS. Como visto anteriormente, isso pode ser feito com a função `BETS.search`. O comando e sua saída são mostrados abaixo.

```
> # Busca em português pela série de produção de bens intermediários
> BETS.search(description = "'bens intermediários'", lang = "pt", view = F)

##          Codes
## 1          1334
## 2         11068
## 3          21864
## 4 ST_1004829
## 5 ST_1007055
## 6 ST_1007057
## 7 ST_1331145
## 8 ST_1331148
##
##                                     Descri
ption
## 1  Indicadores da produção (1991=100) - Por categoria de uso - Bens intermedi
ários
## 2  Indicadores da produção (2002=100) - Por categoria de uso - Bens intermedi
ários
## 3                                     Indicadores da produção (2012=100) - Bens intermedi
ários
## 4 IPA-EX-DI - Bens Intermediários exceto Combustíveis e Lubrificantes para Pro
dução
## 5 IPA-EX-10- Bens Intermediários exceto Combustíveis e Lubrificantes para Pro
dução
## 6 IPA-EX-M - Bens Intermediários exceto Combustíveis e Lubrificantes para Pro
dução
## 7                                     ...BENS INTERMEDI
ÁRIOS
## 8                                     ...BENS INTERMEDI
ÁRIOS
##  Periodicity      start      source      unit
## 1             M  31/01/1975      IBGE  Índice
```



```
## 2      M 31/01/1991      IBGE índice
## 3      M 01/01/2002      IBGE índice
## 4      M      #N/D FGV|IBRE índice
## 5      M      #N/D FGV|IBRE índice
## 6      M      #N/D FGV|IBRE índice
## 7      M 01/12/1997 FGV|IBRE índice
## 8      M 01/12/1997 FGV|IBRE índice
```

Agora, carregamos a série através da função `BETS.get` e guardamos alguns valores para, posteriormente, comparar com as previsões do modelo que será estabelecido. Também criaremos um gráfico (figura 8), pois ele ajuda a formular hipóteses sobre o comportamento do processo estocástico subjacente.

```
> # Obtenção da série de código 21864 (Produção de Bens Intermediários, IBGE)
> data <- BETS.get(21864) > # Gráfico da série
> plot(data, main = "", col = "royalblue", ylab = "PBI (Número Índice)")
> abline(v = seq(2002, 2016, 1), col = "gray60", lty = 3)
```

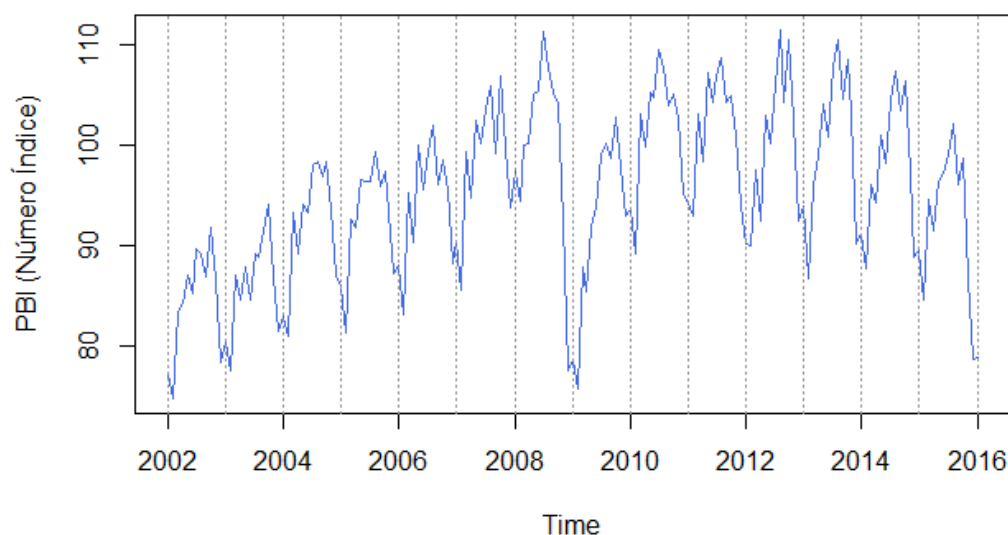


Figura 8: Gráfico da série de produção de bens intermediários no Brasil.

Quatro características ficam evidentes. Primeiramente, trata-se de uma série homocedástica e sazonal na frequência mensal. Este último fato é corroborado pelo gráfico mensal da série (figura 9), que mostra o nível de produção por mês (a média é a linha tracejada)

```
> # Gráfico mensal da série
> monthplot(data, labels = month.abb, lty.base = 2, col = "red", ylab = "PBI (Número Índice)")
```

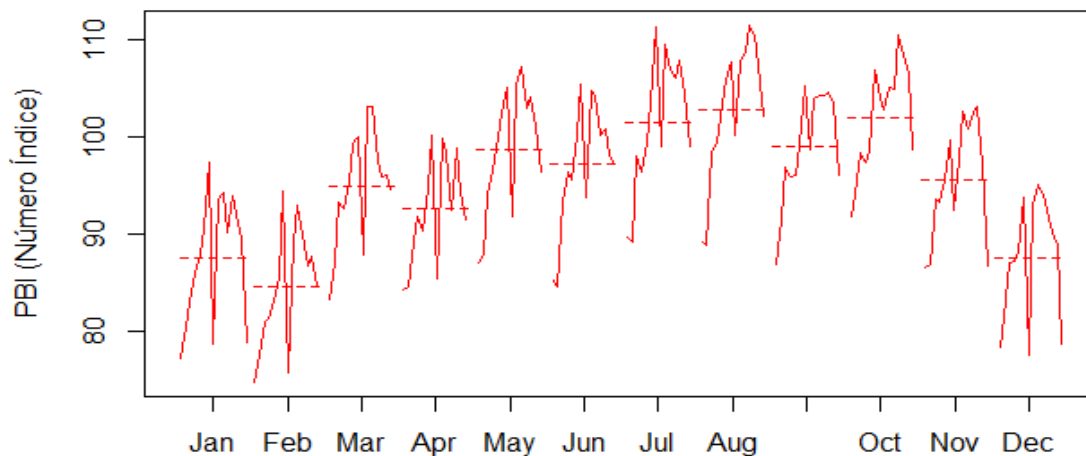


Figura 9: Gráfico mensal da série em estudo.

Um terceiro aspecto marcante da série é a quebra estrutural em novembro de 2008, quando ocorreu a crise financeira internacional e a confiança dos investidores despencou. A quebra impactou diretamente na quarta característica importante da série: a tendência. Inicialmente, a tendência era claramente crescente, mas não explosiva. A partir de novembro de 2008, porém, parece que o nível da série se manteve constante ou até mesmo decresceu. Em um primeiro momento, a quebra estrutural será desconsiderada na estimação dos modelos, mas logo o benefício de levá-la em conta explicitamente ficará claro.

A seguir, criaremos um modelo para a série escolhida de acordo com os passos definidos na anteriormente.

a. Identificação

a.1. Testes para Estacionariedade

Esta subseção trata de um passo crucial na abordagem de Box & Jenkins: a determinação da existência e da quantidade total de raízes unitárias no polinômio autorregressivo não-sazonal e sazonal do modelo. De posse desses resultados, obtemos uma série estacionária através da diferenciação da série original. Assim, poderemos identificar a ordem dos parâmetros através da FAC e FACP, pois isso deve ser feito através de séries estacionárias de segunda ordem.

A função `BETS.ur_test` executa o teste Augmented Dickey Fuller (ADF - [Dickey and Fuller \(1979\)](#)). Ela foi construída em cima da função `ur.df` do pacote `urca` ([Pfaff et al., 2016](#)), que é instalado juntamente com o `BETS`, como explicado na seção 2.3. A vantagem da `BETS.ur_test` é a saída, desenhada para que o usuário visualize rapidamente o resultado do teste e tenha todas as informações de que realmente necessita. Trata-se de um objeto com dois campos: uma tabela mostrando as estatísticas de teste, os valores críticos e se a hipótese nula é rejeitada ou não, e um vetor contendo os resíduos da equação do teste. Esta equação é mostrada abaixo.

$$\Delta y_t = \varphi + \tau_1 t + \tau_2 y_{t-1} + \delta_1 \Delta y_{t-1} + \dots + \delta_{p-1} \Delta y_{t-p+1} + \varepsilon_t \quad (2)$$

As estatísticas de teste da tabela do objeto de saída se referem aos coeficientes φ (média ou drift), τ_1 (tendência determinística) e τ_2 (raiz unitária). A inclusão da média e da tendência determinística é opcional. Para controlar os parâmetros do teste, a `BETS.ur_test` aceita os mesmos parâmetros da `ur.df`, além do nível de significância desejado.

```
> df = BETS.ur_test(y = diff(data), type = "drift", lags = 11,
+                   selectlags = "BIC", level = "1pct")
>
> # Exibir resultado dos testes
> df$results

##      statistic crit.val result
## tau2 -3.024107    -3.46   TRUE
## phi1  4.598267     6.52  FALSE

> # Fazer FAC dos resíduos, com intervalo de confiança de 99%
> BETS.corrrgram(df$residuals, ci=0.99, style="normal", lag.max = 11)
```

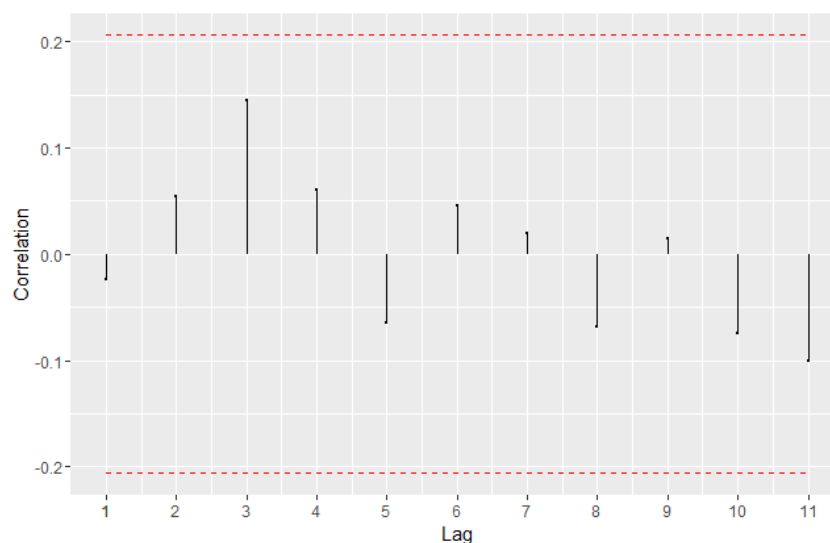


Figura 10: Função de Autocorrelação dos Resíduos do Teste de Dickey-Fuller

Portanto, para a série em nível, observa-se que não se pode rejeitar a hipótese nula de existência de uma raiz unitária ao nível de confiança de 95%, pois a estatística de teste é maior do que o valor crítico. A FAC dos resíduos da equação do teste evidencia que ele foi bem especificado, pois a autocorrelação não é significativa até a décima primeira defasagem (A FAC foi feita com uma função do BETS).

Agora, iremos aplicar a função `diff` à série repetidas vezes e verificar se a série diferenciada possui uma raiz unitária.

```
> ns_roots = 0
> d_ts = diff(data)
>
> # Loop de testes de Dickey-Fuller.
> # A execução é interrompida quando não for possível rejeitar a hipótese nula
> while(df$results[1,"statistic"]> df$results[1,"crit.val"]){
```

Logo, para a série em primeira diferença, rejeita-se a hipótese nula de que há raiz unitária a 5% de significância.

Um outro pacote bastante útil que é instalado com o BETS é o `forecast` (Hyndman, 2015). Usaremos a função `nsdiffs` deste pacote para realizar o teste de Osborn-Chui-Smith-Birchenhall (Osborn et al., 1988) e identificar raízes unitárias na frequência sazonal (em nosso caso, mensal).

```
> library(forecast)
>
> # Testes OCSB para raízes unitárias na frequência sazonal
> nsdiffs(data, test = "ocsb")

## [1] 0
```

Infelizmente, a `nsdiffs` não fornece nenhuma outra informação sobre o resultado do teste além do número de diferenças sazonais que devem ser tiradas para eliminar as raízes unitárias. Para o caso da série em análise, o programa indica que não há raiz unitária mensal, não sendo necessária, portanto, diferenças nesta frequência.

a.2. Funções de Autocorrelação

As conclusões anteriores são corroboradas pela função de autocorrelação da série em primeira diferença (figura 11). Ela mostra que autocorrelações estatisticamente significativas, isto é, fora do intervalo de confiança, não são persistentes para defasagens múltiplas de 12 e no entorno destas, indicando a ausência da raiz unitária sazonal.

As funções do BETS que utilizamos para desenhar correlogramas é a `BETS.corrgram`. Diferentemente de sua principal alternativa, a `Acf` do pacote `forecast`, a `BETS.corrgram` retorna um gráfico atraente e oferece a opção de calcular os intervalos de confiança de acordo com a fórmula proposta

por Bartlett (Bartlett, 1946). Sua maior vantagem, contudo, não pôde ser exibida aqui, pois depende de recursos em flash. Caso o parâmetro `style` seja definido como `'plotly'`, o gráfico torna-se interativo e mostra todos os valores de interesse (autocorrelações, defasagens e intervalos de confiança) com a passagem do mouse, além de oferecer opções de zoom, pannins e para salvar o gráfico no formato png.

```
> # Correlograma de diff(data)
> BETS.corrrgram(diff(data), lag.max = 48, mode = "bartlett", style="normal")
```

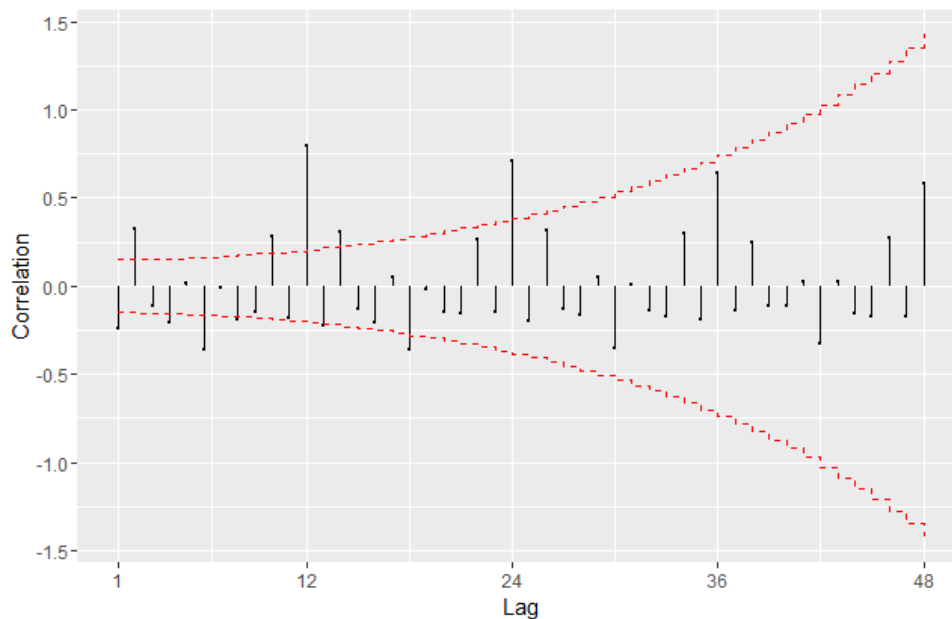


Figura 11: Função de Autocorrelação de ∇Z_t

O correlograma acima ainda não é suficiente para determinamos um modelo para a série. Faremos, então, o gráfico da função de autocorrelação parcial (FACP) de ∇Z_t (definido como a primeira diferença de Z_t). A `BETS.corrrgram` também pode ser utilizada para este fim.

```
> # Função de autocorrelação parcial de diff(data)
> BETS.corrrgram(diff(data), lag.max = 36, type = "partial", style="normal")
```

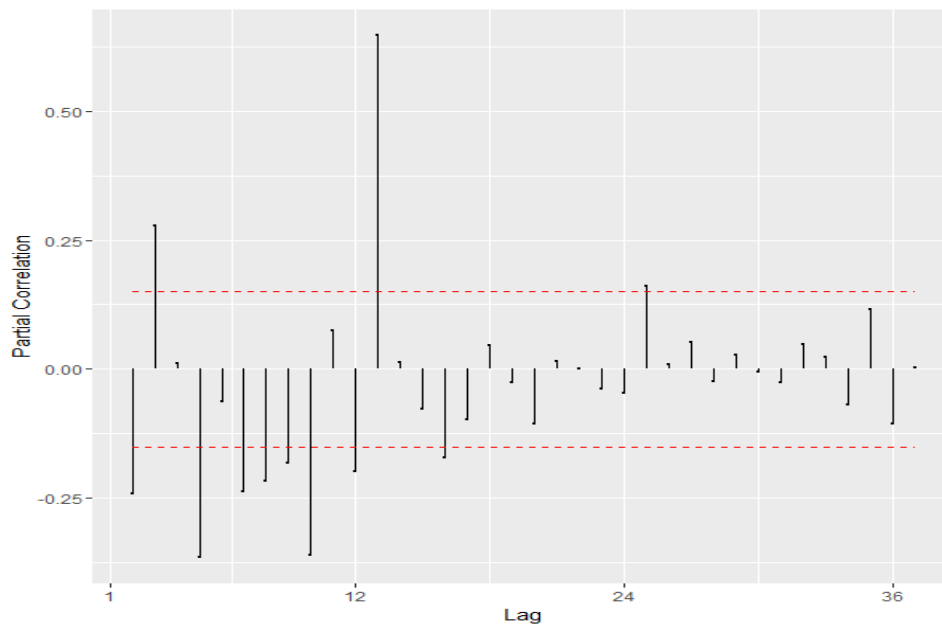


Figura 12: Função de Autocorrelação Parcial de ∇Z_t

A FAC da figura 11 e a FACP da figura 12 podem ter sido geradas por um processo SARIMA(0,0,2)(1,0,0). Esta conjectura se baseia na observação de que as defasagens múltiplas de 12 parecem apresentar corte brusco na FACP a partir da segunda (isto é, a de número 24) e decaimento exponencial na FAC. Além disso, as duas primeiras defasagens da FAC parecem significativas, enquanto as demais, não. Há, ainda, alguma evidência de decaimento exponencial na FACP, exceto na frequência sazonal. Os dois últimos fatos indicam que o polinômio de médias móveis (não sazonal) pode ter ordem 2. Por estas razões, o primeiro modelo proposto para Z_t será um SARIMA(0,1,2)(1,0,0)[12].

b. Estimação

Para estimar os coeficientes do modelo SARIMA(0,1,2)(1,0,0)[12], será aplicada a função Arima do pacote forecast. Os testes t serão feitos através da função BETS.t_test do BETS, que recebe um objeto do tipo arima ou Arima, o número de variáveis exógenas do modelo e o nível de significância desejado, devolvendo um data.frame contendo as informações do teste e do modelo (coeficientes estimados, erros padrão, estatísticas de teste, valores críticos e resultados dos testes).

```
> # Estimacao dos parâmetros do modelo
> model1 = Arima(data, order = c(0,1,2), seasonal = c(1,0,0))
>
> # Teste t com os coeficientes estimados
> # Nível de significância de 1%
> BETS.t_test(model1, alpha = 0.01)
```

##		Coeffs	Std.Errors	t	Crit.Values	Rej.H0
##	ma1	-0.2357448	0.07507782	3.140005	2.606518	TRUE
##	ma2	0.2506416	0.08508626	2.945735	2.606518	TRUE
##	sar1	0.8261948	0.03945109	20.942257	2.606518	TRUE

Conclui-se pela coluna Rej.H0 que os dois coeficientes do modelo, quando estimados por máxima verossimilhança, são estatisticamente significativos a 99% de confiança.

c. Testes de Diagnóstico

O objetivo dos testes de diagnóstico é verificar se o modelo escolhido é adequado. Neste trabalho, duas conhecidas ferramentas serão empregadas: a análise dos resíduos padronizados e o teste de Ljung-Box (Ljung and Box, 1978).

O gráfico dos resíduos padronizados (figura 13) será feito com o auxílio da função `BETS.std_resid`, que foi implementada especificamente para isso.

```
> # Gráfico dos resíduos padronizados
> resid = BETS.std_resid(modell1, alpha = 0.01)
>
> # Evidenciar outlier
> points(2008 + 11/12, resid[84], col = "red")
```

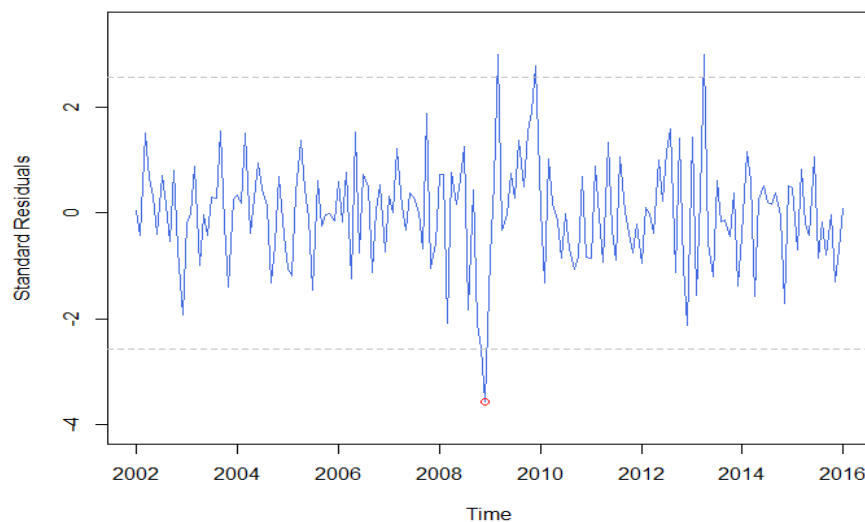


Figura 13: Resíduos padronizados do primeiro modelo proposto

Observamos que há um outlier proeminente e estatisticamente significativo em novembro de 2008. Este ponto corresponde à data da quebra estrutural que identificamos na figura 8. Portanto, foi proposto um segundo modelo, que inclui uma dummy definida como se segue:

$$D_t = \begin{cases} 0, & t < \text{setembro de 2008} \\ 1, & \text{setembro de 2008} \leq t \leq \text{novembro de 2008} \\ 0, & t > \text{novembro de 2008} \end{cases} \quad (3)$$

Esta dummy pode ser criada com a função `BETS.dummy`, como mostramos abaixo. Os parâmetros `start` e `end` indicam o início e o fim do período coberto pela dummy, que nada mais é que uma série temporal cujos valores podem ser apenas 0 ou 1. Os campos `from` e `to` indicam o intervalo em que a dummy deve assumir valor 1.

```
> dummy = BETS.dummy(start = c(2002,1), end = c(2015,10), from = c(2008,9), to = c(2008,11))
> dummy
```

##		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
##	2002	0	0	0	0	0	0	0	0	0	0	0	0
##	2003	0	0	0	0	0	0	0	0	0	0	0	0
##	2004	0	0	0	0	0	0	0	0	0	0	0	0
##	2005	0	0	0	0	0	0	0	0	0	0	0	0
##	2006	0	0	0	0	0	0	0	0	0	0	0	0
##	2007	0	0	0	0	0	0	0	0	0	0	0	0
##	2008	0	0	0	0	0	0	0	0	1	1	1	0
##	2009	0	0	0	0	0	0	0	0	0	0	0	0
##	2010	0	0	0	0	0	0	0	0	0	0	0	0
##	2011	0	0	0	0	0	0	0	0	0	0	0	0
##	2012	0	0	0	0	0	0	0	0	0	0	0	0
##	2013	0	0	0	0	0	0	0	0	0	0	0	0
##	2014	0	0	0	0	0	0	0	0	0	0	0	0
##	2015	0	0	0	0	0	0	0	0	0	0		

Como pode ser visto nos resultados da execução do trecho de código a seguir, a estimação deste modelo através de máxima verossimilhança resultou em coeficientes estatisticamente diferentes de 0 ao nível de significância de 5%, inclusive para a dummy. O gráfico dos resíduos padronizados dos valores ajustados pelo novo modelo (figura 14) também mostra que a inclusão de D_t foi adequada, uma vez que não há mais evidência de quebra estrutural.

```
> # Estimacao dos parâmetros do modelo com a dummy
> model2 = Arima(data, order = c(0,1,2), seasonal = c(1,0,0), xreg = dummy)
>
> # Teste t com os coeficientes estimados
> # Nível de significância de 1%
> BETS.t_test(model2, alpha = 0.01)
```



```
##          Coeffs Std.Errors      t Crit.Values Rej.H0
## ma1    -0.2512842 0.07264346  3.459144    2.606518   TRUE
## ma2     0.3193920 0.09022771  3.539843    2.606518   TRUE
## sar1     0.8361212 0.03822978 21.870938    2.606518   TRUE
## dummy    5.1090010 1.29719620  3.938495    2.606518   TRUE

> # Gráfico dos resíduos padronizados
> resids = BETS.std_resid(model2, alpha = 0.01)
>
> # Evidenciar novembro de 2008
> points(2008 + 11/12, resids[84], col = "red")
```

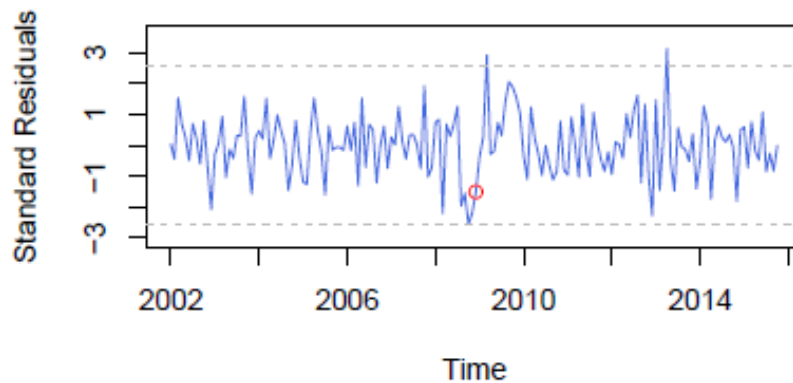


Figura 14: Resíduos padronizados do modelo proposto após a detecção de quebra estrutural

```
> # Mostrar BIC dos dois modelos estimados
> model1$bic

## [1] 847.3404

> model2$bic

## [1] 838.2246
```

Notamos, ainda, que o Bayesian Information Criteria (BIC - [Schwarz \(1978\)](#)) do modelo com a dummy é menor. Logo, também por este critério, o modelo com a dummy deve ser preferido ao anterior.

O teste de Ljung-Box para o modelo escolhido pode ser executado através da função `Box.test` do pacote `stats`. Para confirmar o resultado dos testes, fazemos os correlogramas dos resíduos e vemos se há algum padrão de autocorrelação.

```
> # Teste de Ljung-Box nos resíduos do modelo com a dummy
> boxt = Box.test(resid(model2), type = "Ljung-Box", lag = 11)
> boxt

##
## Box-Ljung test
##
```

```
## data: resid(model2)
## X-squared = 10.867, df = 11, p-value = 0.4545

> # Correlograma dos resíduos do modelo com a dummy
> BETS.corrrgram(resid(model2), lag.max = 20, mode = "bartlett", style = "normal")
```

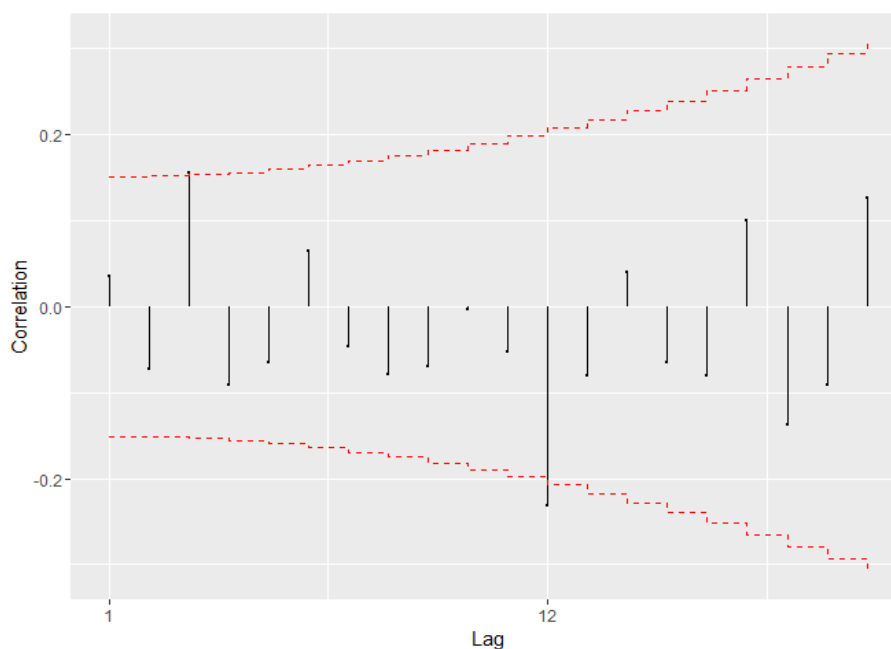


Figura 15: Função de autocorrelação dos resíduos do modelo com a dummy.

O p-valor de 0.4545 indica que há grande probabilidade de a hipótese nula de que não há autocorrelação nos resíduos não seja rejeitada. Parece ser o caso, como mostra a figura 15. Concluimos, então, que o modelo foi bem especificado.

d. Previsões

O BETS fornece uma maneira conveniente para fazer previsões de modelos SARIMA e de GRNNs. A função `BETS.predict` recebe os parâmetros da função `forecast` do pacote homônimo ou da `BETS.grnn.test` (a ser tratada adiante, no segundo estudo de caso) e devolve não apenas os objetos contendo as informações da previsão, mas também um gráfico da série com os valores preditos. Essa visualização é importante para que se tenha uma ideia mais completa da adequação do modelo. Opcionalmente, podem também ser mostrados os valores efetivos do período de previsão.

Fazendo uso da `BETS.predict` para gerar as previsões do modelo proposto. Os parâmetros `object` (objeto do tipo `arima` ou `Arima`), `h` (horizonte de previsão) e `xreg` (a dummy para o período de previsão) são herdados da função `forecast`. Os demais são da própria `BETS.predict`, sendo todos parâmetros do gráfico, com exceção de `actual`, os valores efetivos da série no período de previsão.

```
> new_dummy = BETS.dummy(start = start(data_test), end = end(data_test))
>
> preds = BETS.predict(object = model2, xreg = new_dummy, actual = data_test,
+                       xlim = c(2012, 2016.2), ylab = "Milhões de Reais", style = "normal"
+ )
```

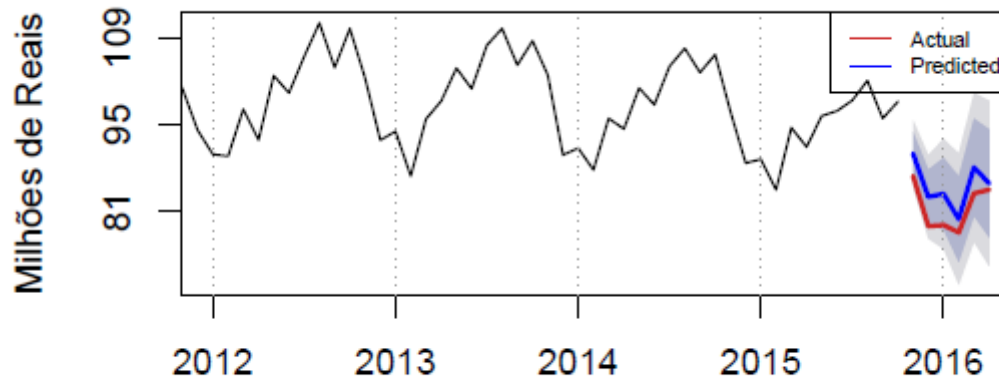


Figura 16: Gráfico das previsões do modelo SARIMA proposto.

As áreas em azul em torno da previsão são os intervalos de confiança de 85% (azul escuro) e 95% (azul claro). Parece que a aderência das previsões foi satisfatória. Para dar mais significado a esta afirmação, podemos verificar várias medidas de ajuste acessando o campo 'accuracy' do objeto retornado.

```
> preds[['accuracy']]

##           ME      RMSE      MAE      MPE      MAPE      ACF1
## Test set -3.453861 3.74175 3.453861 -4.254888 4.254888 -0.2154201
##           Theil's U
## Test set 0.8359621
```

O outro campo deste objeto, 'predictions', contém o objeto retornado pela forecast (ou pela BETS.grnn.test, se for o caso). Na realidade, este campo ainda conta com um dado adicional: os erros de previsão, caso sejam fornecidos os valores efetivos da série no período de previsão.

O uso da BETS.report para a modelagem SARIMA

A função BETS.report executa toda a modelagem Box & Jenkins para qualquer série presente no banco de dados do BETS e gera um relatório com os resultados, como foi dito no início desta seção. Ela aceita três parâmetros, descritos na tabela [3](#), abaixo.

Nome	Tipo	Descrição
ts	integer	Código da série no banco de dados do BETS.
mode	character	Tipo da modelagem a ser efetuada. Atualmente, pode ser SARIMA ou GRNN
parameters	list	Parâmetros do relatório
type	character	Modo de exibição. Por enquanto, apenas o valor default, 'educational', é válido. Significa que comentários didáticos acompanharão os gráficos e trechos de código.

Tabela 3: Parâmetros da função BETS.report

Se utilizarmos a modelagem SARIMA, a lista params deverá ser composta dos seguintes campos:

Nome	Tipo	Descrição
lag.max	integer	Defasagem máxima dos correlogramas
n.ahead	integer	Números de passos à frente nas previsões

Tabela 4: Campos da lista params da função BETS.report caso a análise seja do tipo SARIMA

Para modelar a série de produção de bens intermediários de maneira bastante similar a que fizemos neste artigo, não é necessário sequer carregar os dados, bastando executar o trecho de código que mostramos a seguir.

```
> parameters = list(
+   lag.max = 48,
+   n.ahead = 12 )
>
> BETS.report(ts = 21864, params = parameters)
```

O resultado abre automaticamente, na forma de um arquivo .html. Um trecho deste arquivo pode ser visto na figura [17](#). Ele contém muito mais do que pode ser visto na figura, a saber:

- As informações da série tal como se encontram na tabela de metadados do BETS. O gráfico da série, feito com o pacote dygraphs ([Vanderkam et al., 2016](#)).
- Os passos envolvidos na identificação de um possível modelo: testes de raiz unitária (por enquanto, apenas os testes ADF e o OCSB) e correlogramas da série original, da série diferenciada (se for o caso) e sazonalmente diferenciada (se for o caso).

- A estimação dos parâmetros e o resultado da seleção automática do modelo pela função `auto.arima` do pacote `forecast`.
- Teste de Ljung-Box para autocorrelação nos resíduos.
- As previsões n passos à frente utilizando a função `forecast` e um gráfico da série original com os valores previstos e os intervalos de confiança, também feito com o `dygraphs`

3.3.2 General Regression Neural Networks (GRNNs)

A `BETS.report` também produz relatórios que aplicam a metodologia de redes neurais. Em particular, de redes do tipo GRNN (General Regression Neural Network), tal como proposta por [Specht \(1991\)](#). O treinamento e as previsões das redes são feitas com o auxílio do pacote `grnn` ([Chasset, 2013](#)). Neste método, outras séries podem ser utilizadas como regressoras, isto é, como variáveis explicativas.

A figura [18](#) ilustra a estrutura de uma GRNN. A primeira camada, chamada camada de distribuição, é apenas uma representação dos vetores de entrada. Na segunda camada, há um neurônio para cada padrão de treinamento. Suas funções de ativação são gaussianas multivariadas centradas no correspondente vetor de treinamento. A camada seguinte possui um neurônio que calcula o numerador da equação geral da GRNN (equação [4](#)), onde os pesos são os próprios vetores de treinamento. O segundo neurônio desta camada calcula o denominador da equação e utiliza pesos de valor 1. A função do denominador é normalizar a saída.

$$Y(Z) = \frac{\sum_{i=1}^n X_i \exp\left(\frac{D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^n \exp\left(\frac{D_i^2}{2\sigma^2}\right)} \quad (4)$$

onde $D^2 = (Z - x_i)^T (Z - x_i)$ e tal que X_i é o vetor de treinamento i , $i \in [1, n]$. Z é um vetor de entrada, $Y(Z)$ é a saída e σ é o campo receptivo dos neurônios.

Fitted SARIMA Model

Talitha Speranza

2016-08-22

User-Defined Parameters

Parameter	Value	Variable
Series Code	21864	ts
Maximum Lag	48	lag.max
Prevision Horizon	12	n.ahead

Getting the Time Series from the BETS database

```
library(BETS)
data = BETS.get(ts)
```

Information About the Series

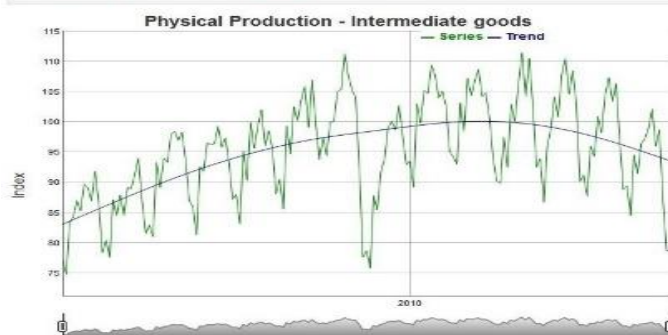
```
info <- BETS.search(code = ts, view = F)
```

Code	Description	Periodicity	Start	Source	Unit
21864	Physical Production - Intermediate goods	M	2002.1	IBGE	Index

Graph

```
library(mFilter)
trend = fitted(hpFilter(data))

library(dygraphs)
dygraph(cbind(Series = data, trend = trend), main = info[, "Description"]) %>%
  dyRangeSelector(strokeColor = "gray", fillColor = "gray") %>%
  dyAxis("y", label = info[, "Unit"])
```



Unit Root Tests

Augmented Dickey-Fuller

```
df = BETS.ur_test(y = diff(data), type = "drift", lags = 11, selectlags = "BIC", level = "5pct")
df$results
```

```
##      statistic crit.val result
## tau2 -2.734365   -2.88  TRUE
## phi1  3.803556    4.63 FALSE
```

For a 95% confidence interval, the test statistic `tau3` is greater than the critical value. We therefore conclude that there must be a unit root. Now, we are going to repeatedly apply `diff` to the series and check if the differenced series has a unit root.

```
ns_roots = 0
d_ts = data

while(df$results[1, "statistic"] > df$results[1, "crit.val"]){
  ns_roots = ns_roots + 1
  d_ts = diff(d_ts)
  df = BETS.ur_test(y = d_ts, type = "none", lags = 6, selectlags = "BIC", level = "5pct")
  print(df$results)
}
```

```
##      statistic crit.val result
## tau1 -8.707792   -1.95 FALSE
```

These tests found that there must be a total of 1 unit root(s)

Osborn-Chui-Smith-Birchenhall

This test will be performed for lag 12, that is, the frequency of the series 21864.

```
library(forecast)
s_roots = nsdiffs(data)
print(s_roots)
```

Figura 17-2: Início do arquivo de saída da função `BETS.report` para a série em estudo.

Nota-se que quanto mais distante o vetor de entrada estiver do padrão de treinamento de determinado neurônio, mais baixa será sua contribuição para o termo correspondente nos somatórios. É importante observar que a performance melhora consideravelmente se os vetores de entrada forem normalizados, pois isso evita distorções causadas por diferentes escalas ou valores discrepantes.

Como mencionado, há apenas um parâmetro a ser definido na GRNN. Este parâmetro é o campo receptivo, σ (sigma), o desvio padrão das funções de ativação.

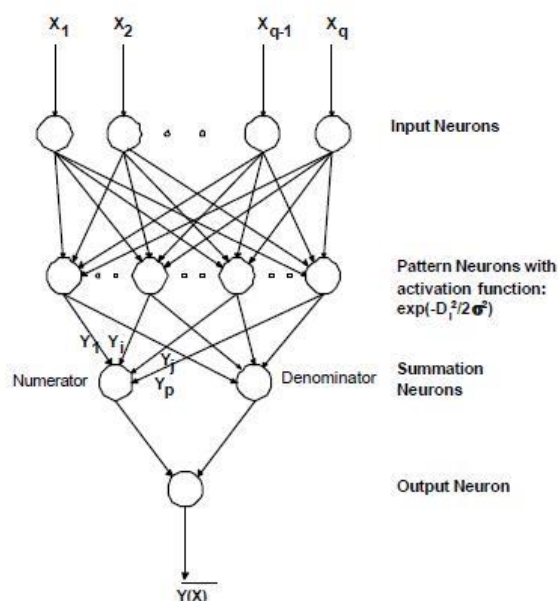


Figura 18: Esquema de uma GRNN

A seguir, a análise, como mencionado anteriormente, da série do Índice de Preços ao Consumidor Amplo (IPCA) a partir de uma GRNN, usando ferramentas providas pelo BETS.

Preliminares

Pretende-se modelar a série do IPCA a partir da noção da Curva de Phillips ([Phillips, 1958](#)). Uma versão da curva é a representada pela equação 5. Ela busca explicar como a taxa de inflação efetiva é formada. A hipótese básica é que ela é proporcional a uma combinação linear da inflação, da meta da inflação e do hiato do produto, todos avaliados no período anterior.

$$\pi_t = \alpha \pi_{t-1} + (1 - \alpha) \pi^*_{t-1} + \beta \hat{y}_{t-1} \quad (5)$$

onde

π é a taxa de inflação

α é a persistência da taxa de inflação

β é a elasticidade da inflação em relação ao hiato do produto

\hat{y} é o hiato do produto

No entanto, há indícios de que modelos lineares não captam muito bem a relação entre estas variáveis. Felizmente, problemas não lineares podem ser facilmente resolvidos com o uso de redes neurais artificiais, como faremos aqui.

Todas as séries de que necessitamos estão presentes nas bases do BETS, bastando utilizar a BETS.get para obtê-las. A série das metas de inflação medida pelo IPCA é anual e termina em 2015, mas como as demais são mensais e terminam em fevereiro de 2016, devemos efetuar uma transformação para tornar os dados mensais.

```
> target_monthly = vector(mode = "numeric")
>
> for(t in target){
+   target_monthly = c(target_monthly, rep(t,12))
+ }
>
> target = ts(target_monthly, start = c(1999,1), end = c(2015,12), frequency = 12
)
```

A série do PIB também deve ser transformada. Deve-se trabalhar com os valores reais do hiato do produto, ou seja, teremos que deflacionar a série do PIB utilizando algum índice de inflação e calcular seu desvio em relação à tendência observada. A primeira operação será realizada com uma função do BETS, a BETS.deflate. O deflator selecionado foi o próprio IPCA. O argumento type indica que a série do IPCA está em ponto percentual (também é possível deflacionar com séries que estejam em porcentagem ou em número índice).

```
> gdp_real = BETS.deflate(gdp, ipca, type = "point.perc")
```

A tendência será encontrada através de um filtro HP (Hodrick and Prescott, 1997). Este filtro está disponível no pacote mFilter (Balcilar, 2016).

```
> library(mFilter)
>
> trend = fitted(hpfilter(gdp_real))
> h_gdp_real = gdp_real - trend
```

Por fim, será construído um gráfico das duas principais séries que serão utilizadas daqui pra frente: o IPCA e o hiato do PIB real. Mostraremos apenas os dados após 1996, uma vez que o Plano Real, de

1994, resultou em uma quebra estrutural muito significativa e os valores discrepantes entre os dois períodos dificultam a visualização.

```
> h_gdp_real = window(h_gdp_real, start = c(1996,1))
> ipca = window(ipca, start = c(1996,1))
>
> plot(h_gdp_real, type="l", col="firebrick3", ylab = "Milhões de Reais")
> par(new=TRUE)
> plot(ipca, type="l", col="royalblue", xaxt="n", yaxt="n", xlab="", ylab="")
> axis(4)
> mtext("%", side=4, line=3)
> legend("topright", col=c("firebrick3", "royalblue"),
+       lty=1, legend=c("Hiato do PIB Real", "IPCA"), cex = 0.7)
```

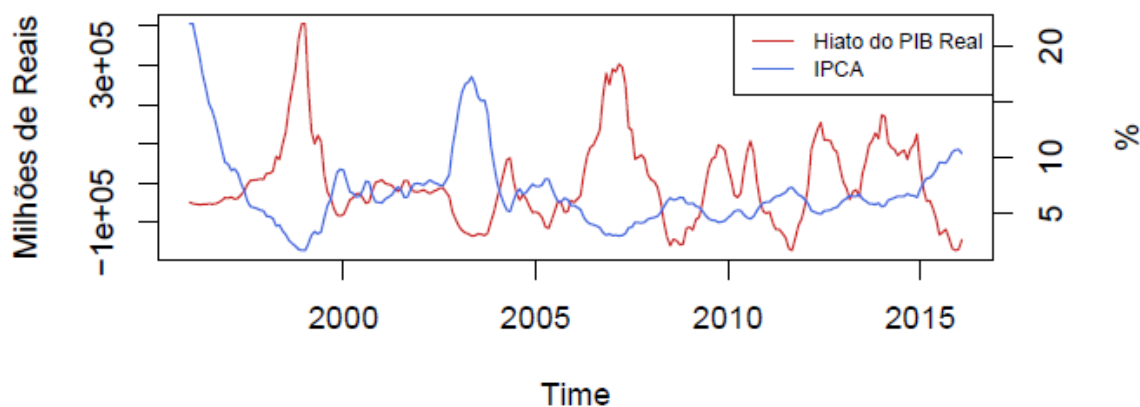


Figura 19: Gráfico do IPCA e do Hiato do PIB Real

Observa-se que há uma nítida relação, embora complexa, entre as duas séries, o que sugere que o hiato do PIB real de fato tem poder explicativo sobre o IPCA. Enxergamos uma relação pouco intuitiva, pois aparenta ser negativa. Isso, evidentemente, ocorre por haver uma defasagem do impacto do hiato na inflação.

A relação da meta da inflação com a inflação efetiva é mais difícil de notar, já que as metas variam muito pouco. Por essa razão, não chegamos a exibir um gráfico mostrando as duas séries. Entretanto, sabemos o efeito que as metas têm sobre a formação das expectativas de inflação e, então, não podemos ignorá-las em nossa análise.

Normalização

A normalização dos valores de entrada é um passo muito importante se estivermos trabalhando com redes neurais. Normalizar significa padronizar os valores das séries de modo a suavizar sua variabilidade e aumentar a acurácia da computação numérica, removendo redundâncias. O efeito da normalização, portanto, é melhorar a performance das redes, ajudando a evitar falhas de simulação e tornando-as mais eficientes.

Os procedimentos de normalização foram considerados relevantes o suficiente para que fosse criada uma função específica que os executasse. Esta função é a `BETS.normalize`. Sua estrutura é bastante simples e há apenas três parâmetros: os dados que devem ser padronizados, o tipo de normalização e o intervalo em que os valores devem ficar. O segundo parâmetro pode receber os valores 'maxmin' ou 'scale'. O terceiro só faz sentido se for utilizada a normalização 'maxmin' e seu valor default é `c(0,1)`.

Aqui, será aplicado o outro tipo de normalização disponível, o `scale`, que realiza duas operações: subtrai de cada elemento a média da série e os divide pelo desvio padrão. Vamos, também, guardar a média e o desvio padrão da série do IPCA, de modo a posteriormente obter os valores preditos pela GRNN na escala original.

```
> # Normalização dos valores das séries
> h_gdp_real.norm = BETS.normalize(h_gdp_real, mode = "scale")
> ipca.norm = BETS.normalize(ipca, mode = "scale")
> target.norm = BETS.normalize(target, mode = "scale")
>
> ipca.mean = mean(ipca)
> ipca.sd = sd(ipca)
```

Treinamento

Antes de treinarmos a rede, é necessário inicializar os parâmetros aceitos pela função de treinamento, a `BETS.grnn.train`. Ela tem o seguinte protótipo:

```
BETS.grnn.train = function(train.set, sigma, step = 0.1, select =
                           TRUE, criterion = "MAPE")
```

O argumento `train.set` é uma lista de objetos do tipo `ts` (séries temporais), onde a primeira deve ser a variável dependente (no nosso caso, o IPCA) e as demais, os regressores. Cada defasagem deve ser

fornecida como um regressor adicional. Trabalharemos com as primeiras duas defasagens do IPCA (que daremos o nome de 'ipca_1' e 'ipca_2'), do hiato do produto ('h_gdp_1' e 'h_gdp_2') e da meta de inflação ('target_1' e 'target_2'). Se o valor do parâmetro select for mantido como o default (TRUE), cada possível combinação de regressores formará uma rede a ser treinada e avaliada de acordo com seus valores ajustados. A medida de ajuste pode especificada pelo argumento criterion, cujo valor padrão é o MAPE. Caso select seja FALSE, apenas uma rede será treinada, utilizando todos os regressores da lista.

Os campos sigma e step se referem ao parâmetro σ , o campo receptivo dos neurônios. O valor de sigma pode ser único ou um intervalo (um vetor com dois números). Caso seja um intervalo, σ será variado (sigma[2]-sigma[1])/step vezes, sendo portanto o tamanho da variação determinado por step (por padrão, 0.1). Isto significa que, para cada conjunto de regressores, serão treinadas tantas redes quanto forem os valores de sigma. Novamente, o melhor resultado será definido de acordo com critério estabelecido pelo parâmetro criterion.

Como a série da meta de inflação começa em janeiro de 1999 e usaremos duas defasagens, trabalharemos apenas com janelas de março de 1999 a fevereiro de 2015, para todas as séries. Os demais dados serão aproveitados para fazer previsões, na fase de teste das redes.

```
> # Prepara uma lista com as séries originais
> series = vector(mode = "list")
> series[[1]] = ipca.norm
> series[[2]] = h_gdp_real.norm
> series[[3]] = target.norm
>
> # Cria a lista que conterá as séries originais e as defasadas
> complete = vector(mode = "list")
> complete[[1]] = ipca.norm
>
> # Determina o número máximo de defasagens (2)
> lag.max = 2

>
> # Preenche a lista com as defasagens
> nvars = length(series)
>
> for(i in 1:nvars){
+   s = 1 + (i-1)*lag.max
+   for(j in 1:lag.max){
+     complete[[s + j]] = lag(series[[i]],-j)
+   }
+ }
>
```

```

> # Redimensiona e nomeia as séries, colocando-as no conjunto de treinamento
> train = vector(mode = "list")
> names = c("ipca", "ipca_1", "ipca_2", "h_gdp_1", "h_gdp_2", "target_1", "target_2")
>
> for(i in 1:length(complete)){
+   train[[i]] = window(complete[[i]], start = c(1999,3), end = c(2015,2))
+ }

```

Finalmente, a `BETS.grnn.train` é acionada. A função retorna uma lista de até 20 resultados, cada um representando uma rede treinada e suas informações. Eles são ordenados de acordo com a medida de ajuste, do melhor resultado para o pior. O objeto que representa um resultado possui vários campos, descritos na tabela [5](#).

```

> results = BETS.grnn.train(train, sigma = c(0.1,1.6), step = 0.1)

## [1] "GRNN: List of Trained Networks with Best Performance"
##      mape      regs sigma
## 63 11.76231 2,3,4,5,6,7 0.1
## 57 12.22237  2,3,4,5,6  0.1
## 58 12.44523  2,3,4,5,7  0.1
## 61 16.99125  2,4,5,6,7  0.1
## 48 17.48345   2,4,5,6   0.1
## 62 17.83007  3,4,5,6,7  0.1
## 49 17.96930   2,4,5,7   0.1
## 42 18.42746   2,3,4,5   0.1

```

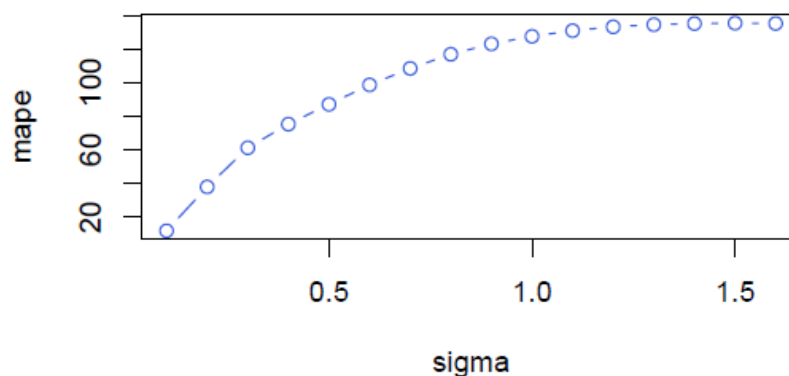


Figura 20: Gráfico das previsões da GRNN

Nota-se que o valor do MAPE aumenta quase que exponencialmente com o aumento do campo receptivo σ , com um aparente limite superior de 140. O gráfico sugere, ainda, que o limite inferior do intervalo para a variação do σ seja diminuído, de forma a tornar o MAPE do ajuste menor.

Entretanto, esbarramos em um problema: tipicamente, valores muito baixos de σ costumam levar ao overfitting (supertreinamento, isto é, a rede incorpora o ruído ao seu aprendizado). Na prática, isso significa que as previsões serão piores, mesmo que o MAPE dos valores ajustados seja muito baixo, ou seja, a rede não aprende a generalizar.

Nome	Tipo	Descrição
accuracy	numeric	Valor da medida de desempenho da rede, calculada através dos valores ajustados e dos valores efetivos da série em estudo. Por default, o valor do MAPE.
fitted	vector	Valores ajustados pela rede, isto é, valores previstos pela rede um passo a frente, após o treinamento e utilizando como entradas os valores dos regressores.
net	list	Objeto retornado pela função <code>grnn</code> , representando uma rede treinada.
sigma	numeric	Parâmetro σ
regressors	vector	Índices dos regressores no conjunto de treinamento original
sigma.accuracy	data.frame	Medida de ajuste versus sigma de cada rede treinada com o mesmos regressores, mas variando o σ .
residuals	vector	Resíduos, ou seja, valores efetivos menos valores ajustados.

Tabela 5: Campos da lista de resultados retornados pela função `BETS.grnn.train`

Na tabela acima, fica evidente que o uso da `BETS.grnn.train` traz muitas vantagens em relação à função de treinamento do pacote `grnn`. Não apenas os regressores podem ser automaticamente escolhidos, mas o parâmetro σ pode ser variado e, ao final, são fornecidas as mais diversas informações sobre os resultados do treinamento das melhores redes. Além disso, a função retorna mais dois objetos: um gráfico mostrando a evolução do desempenho contra o sigma das redes que utilizam o conjunto de regressores que resultou na melhor performance e uma tabela resumindo os resultados.

Testes

Há também uma função que executa os testes das redes separadamente, a `BETS.grnn.test`. Ela recebe a lista de resultados gerada pela função de treinamento e uma lista de séries de entrada, onde a primeira série deve conter os valores efetivos da variável dependente e as demais, os valores dos regressores no período de previsão. É importante que a ordem dada às séries desta lista siga a ordem da lista que representou o conjunto de treinamento. O uso da `BETS.grnn.test` é exemplificado abaixo.

```
> # Preparação da lista de séries
> test = vector(mode = "list")
>
```

```

> for(i in 1:length(complete)){
+   test[[i]] = window(complete[[i]], start = c(2015,3), end = c(2015,11))
+ }
>
> # Testes
> best.net = BETS.grnn.test(results,test)
>
> # Campo 'accuracy' do objeto best.net (MAPE)
> best.net[['accuracy']]

## [1] 36.70073

> # Regressores da melhor rede em termos de previsão
> regs = best.net[['regressors']]
> names[regs]

## [1] "ipca_1" "ipca_2" "h_gdp_1"

```

É interessante notar que os regressores escolhidos incluem o próprio IPCA e a primeira defasagem do hiato do PIB real, como previsto pela Curva de Phillips. Entretanto, a meta da inflação parece não ter muito poder explicativo, ao contrário do que esperávamos. Este pode ser um indicativo, ainda que muito tênue, de que os comunicados do Banco Central não são tidos como críveis pelo mercado, de maneira geral, mas o modelo é demasiado simples para que cheguemos a alguma conclusão deste tipo.

A `BETS.grnn.test` retorna uma lista com informações sobre a melhor rede em termos do critério de avaliação das previsões. Por padrão, esse critério é novamente o MAPE, calculado a partir dos valores previstos e efetivos. A lista retornada é compatível com a esperada pela `BETS.predict` e, portanto, também poderíamos utilizá-la para efetuar as previsões. Para isso, passamos os mesmos parâmetros da `BETS.grnn.test`, tal como fizemos no caso da modelagem SARIMA.

Assim, é possível visualizar o gráfico das previsões, com uma vantagem adicional: a `BETS.predict` permite que os valores sejam desnormalizados, bastando que a média e o desvio padrão sejam fornecidos através do parâmetro `unnorm`. Isto facilita a apreciação dos valores e permite que um novo cálculo da medida de ajuste seja feito. A medida calculada com os valores desnormalizados é mais significativa, pois representa os desvios na escala em que os valores foram de fato observados.

```

> preds = BETS.predict(results, test, actual = test[[1]],
+                      unnorm = c(ipca.mean, ipca.sd), xlim = c(2012, 2015 + 11/1
+                      2),
+                      ylab = "%", style = "normal")
> preds[['accuracy']]

##               ME          RMSE          MAE          MPE          MAPE          ACF1
## Test set 0.8942656 1.096747 0.9128709 9.266343 9.486004 0.6560495

```

```
##           Theil's U
## Test set  2.808702
```

Como foi avaliado se o MAPE de 9.49 é um resultado satisfatório? Uma boa estratégia é comparar com o MAPE de um modelo linear. Isto pode ser feito rapidamente através da função `auto.arima` do pacote `forecast`. Ela busca selecionar automaticamente o melhor modelo SARIMA para uma determinada série. Nem sempre o melhor resultado possível é encontrado, mas é uma boa maneira de se ter uma ideia de qual seria um modelo linear razoável.

```
> model = auto.arima(train[[1]])
>
> bm = BETS.predict(model, h=9, actual = test[[1]],
+                   unnorm = c(ipca.mean, ipca.sd), style = "none")
>
> bm[['accuracy']]

##           ME           RMSE           MAE           MPE           MAPE           ACF1
## Test set  0.8445419  0.9588219  0.8445419  8.853872  8.853872  0.5885077
##           Theil's U
## Test set  2.486728
```

Se consultarmos o campo `arma` do objeto retornado pela `auto.arima`, vemos que o modelo selecionado foi um SARIMA(1,1,0)(2,0,2)[12], com um MAPE de 8.85. É um MAPE um pouco menor que os das previsões da GRNN, mas antes de fazermos qualquer afirmação sobre a precisão relativa das previsões dos dois modelos, precisamos estabelecer se elas são estatisticamente diferentes entre si. Isto pode ser feito através do teste do sinal ([Lehmann, 1998](#)), definido como se segue.

$$sign = \frac{E[S_n] - 0.5N}{0.5\sqrt{N}} \sim N(0,1) \quad (6)$$

$$H_0: E[S_n] = 0.5N, H_1: E[S_n] > 0.5N$$

onde

- `sign` é a estatística de teste;
- $E[S_n]$ é o valor esperado do número de vezes em que os erros de um dado modelo são maiores que os erros do outro modelo;
- N é tamanho do vetor de erros.

O código para executar o teste do sinal com os erros dos nossos modelos seria:

```
> errors.sarima = bm[[2]]$forecasting.errors
> errors.grnn = preds[[2]]$forecasting.errors
```

```

>
> dif = errors.sarima - errors.grnn
> E = sum(sapply(dif, function(x){return(x > 0)}))
> mean = 0.5*9
> std = 0.5*sqrt(9)
>
> sign = (E - mean)/std
> p.value = pnorm(sign)

```

O p-valor de $p.value = 0.37$ indica que a hipótese nula de igualdade dos erros não seria rejeitada a níveis de significância bastante altos, o que nos leva a crer que não há diferenças estatísticas entre as previsões dos dois modelos.

No entanto, o modelo linear apresenta algumas desvantagens. Como é um SARIMA(1,1,0)(2,0,2)[12], há 7 variáveis na equação do modelo, contrastando com o número menor de regressores da GRNN, apenas 3. Ademais, não foi necessário adotar nenhuma hipótese sobre o comportamento do IPCA, além de sua dependência do hiato do PIB, para construir a rede neural. Tal não é o caso dos modelos SARIMA. Além da óbvia restrição de linearidade, deve-se formular suposições sobre vários outros aspectos da série em estudo, como a sazonalidade, a tendência e os ciclos.

O uso da BETS.report para a modelagem através de uma GRNN

Para a modelagem através de uma GRNN, a lista de parâmetros que é argumento da BETS.report deve possuir os campos apresentados abaixo (tabela 6).

Nome	Tipo	Descrição
code	integer	Código da série que funcionará como variável dependente (a série que é objeto de estudo).
auto.reg	boolean	Indica se a própria variável dependente deve ser usada como regressor.
present.regs	boolean	Indica se os valores presentes dos regressores devem entrar como variáveis explicativas.
lag.max	integer	Defasagem máxima dos regressores.
regs	integer	Vetor de códigos das séries que devem funcionar como variáveis explicativas. As defasagens de cada uma serão escolhidas por busca exaustiva e apresentadas no relatório.
start.train	integer	Período de início do conjunto de treinamento.
end.train	integer	Período de fim do conjunto de treinamento.
start.test	integer	Período de início do conjunto de testes.
end.test	integer	Período de fim do conjunto de testes.
sigma.interval	numeric	Intervalo em que o único parâmetro da GRNN, o σ (sigma), deve ser variado. Se apenas um valor for fornecido, σ não será variado.
sigma.step	numeric	Tamanho da variação do parâmetro σ .

Tabela 6: Campos da lista params da função BETS.report caso a análise seja do tipo GRNN

Lembramos que a utilização da BETS.report foi explicada anteriormente nesta seção, quando descrevemos seu uso no caso de modelos SARIMA.

Futuramente, serão ampliados os relatórios dinâmicos, dando ao usuário novas opções. Uma delas já está prevista na estrutura da função: o modo de visualização que não inclui explicações sobre a metodologia, assumindo tom mais técnico e sucinto. Além de desenvolver novos métodos de modelagem, como Holt-Winters, lógica fuzzy, Box & Jenkins com funções de transferência e outras arquiteturas de redes neurais, planejamos refinar as análises em si.

No relatório SARIMA seria interessante, por exemplo, implementar outros testes de raiz unitária bem estabelecidos, como o KPSS ([Kwiatkowski et al., 1992](#)) e o Phillips-Perro, ([Phillips and Perron, 1988](#)), e testes de diagnóstico como o de Breusch–Godfrey para autocorrelação nos resíduos ([Breusch, 1978](#); [Godfrey, 1978](#)) e o de Chow para quebras estruturais ([Chow, 1960](#)). Outra funcionalidade sem dúvida proveitosa seria a de decomposição das séries, que poderia ser feita através do seasonal ([Sax, 2016](#)), um pacote que oferece uma interface para o X-13-ARIMA-SEATS ([Bureau, 2015](#)).

O pacote BETS será submetido ao CRAN (repositório oficial do R) tornando ele mais acessível devido ao fato de que muitos não conhecem o github. Ao longo do capítulo foi mencionado muitos aprimoramentos para futuras versões, aprimoramentos de funções e otimização das estruturas, foram mencionado o servidor independente para o BETS facilitando o processo de atualização da base de dados sem a necessidade de submeter uma nova versão. Algo interessante para a parte de gerenciamento de dados do pacote, será a criação de um addin, uma interface que torna o processo de pesquisa e obtenção de séries temporais mais confortável. Um sistema de login para que o usuário que possui assinatura das séries da FGV possa acessá-las diretamente pelo R

Bibliografia

M. Balciilar. mfilter: Miscellaneous time series filters, 2016. URL <http://cran.r-project.org/package=mfilter>. [p22]

M. S. Bartlett. On the theoretical specification and sampling properties of autocorrelated time series.

Supplement to the Journal of the Royal Statistical Society, 1946. [p14]

R. Bivand, V. J. Carey, S. DebRoy, S. Eglen, R. Guha, N. Lewin-Koh, M. Myatt, B. Pfaff, B. Quistorff, F. Warmerdam, and S. Weigand. foreign: Technical Trading Rules, 2016. URL <http://cran.r-project.org/package=foreign>. [p4]

G. E. P. Box and G. M. Jenkins. Time Series Analysis forecasting and control. Holden Day, San Francisco, 1970. [p10]

T. S. Breusch. Testing for autocorrelation in dynamic linear models. Australian Economic Papers, 1978. [p28]

U. C. Bureau. X13-arma-seats reference manual accessible html output version. 2015. URL <https://www.census.gov/ts/x13as/docX13AS.pdf>. [p29]

K. S. Chan and B. Ripley. TSA: Time Series Analysis, 2012. URL <http://CRAN.R-project.org/package=TSA>. [p4]

P. Chasset. grnn: General regression neural network, 2013. URL <http://cran.r-project.org/package=grnn>. [p19]

G. C. Chow. Tests of equality between sets of coefficients in two linear regressions. Econometrica, 1960. [p29]

D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. Journal of the American Statistical Association, 74, 1979. [p12]

A. A. Dragulescu. xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files, 2016. URL <http://cran.r-project.org/package=xlsx>. [p4]

M. Gagolewski and B. Tartanus. stringi: Character String Processing Facilities, 2016. URL <http://cran.r-project.org/package=stringi>. [p4]

I. Gavrilov and R. Pusev. normtest: Tests for Normality, 2016. URL <http://cran.r-project.org/package=normtest>. [p4]

A. Ghalanos. rugarch: Univariate GARCH Models, 2016. URL <http://cran.r-project.org/package=rugarch>. [p4]

L. G. Godfrey. Testing against general autoregressive and moving average error models when the regressors include lagged dependent variables. Econometrica, 1978. [p29]

S. Graves. FinTS: Companion to Tsay (2005) Analysis of Financial Time Series, 2014. URL <http://cran.r-project.org/package=FinTS>. [p4]

G. Grothendieck. sqldf: Perform SQL Selects on R Data Frames, 2015. URL <https://cran.r-project.org/web/packages/sqldf/index.html>. [p2, 4]

R. J. Hodrick and E. C. Prescott. Postwar u.s. business cycles: An empirical investigation. Journal of Money, Credit, and Banking, 1997. [p22]

T. Hothorn, A. Zeileis, R. W. Farebrother, C. Cummins, G. Millo, and D. Mitchell. lmtest: Testing Linear Regression Models, 2016. URL <http://cran.r-project.org/package=lmtest>. [p4]

R. J. Hyndman. forecast: Forecasting Functions for Time Series and Linear Models, 2015. URL <http://cran.r-project.org/package=forecast>. [p4, 13]

D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root. Journal of Econometrics, 1992. [p28]

D. T. Lang. SSOAP: Client-side SOAP access for S, 2012a. URL <https://cran.rstudio.com/src/contrib/Archive/SSOAP/>. [p1]

D. T. Lang. XMLSchema: R facilities to read XML schema, 2012b. URL <https://cran.r-project.org/>

[src/contrib/Archive/XMLSchema/](#). [p1]

E. L. Lehmann. Nonparametrics: Statistical methods based on ranks. Prentice Hall, Upper Saddle River, NJ, 1998. [p27]

G. M. Ljung and G. E. P. Box. On a measure of a lack of fit in time series models. *Biometrika*, 65, 1978. [p15]

D. C. Montgomery, C. L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. Wiley, Arizona, 2015. [p10]

D. Osborn, A. Chui, J. Smith, and C. Birchenhall. Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics*, 1988. [p13]

B. Pfaff, E. Zivot, and M. Stigler. *urca: Unit Root and Cointegration Tests for Time Series Data*, 2016. URL <http://cran.r-project.org/package=urca>. [p4, 12]

A. W. Phillips. The relationship between unemployment and the rate of change of money wages in the united kingdom 1861-1957. *Economica*, 1958. [p21]

P. C. Phillips and P. Perron. Testing for a unit root in time series regression. *Biometrika*, 1988. [p28]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL <http://www.R-project.org/>. ISBN 3-900051-07-0. [p1]

A. Rademaker. R package ssoap. <http://arademake.github.io/blog/2012/01/02/package-SSOAP.html>, 2012. Accessed: 2016-07-27. [p1]

C. Sax. seasonal: R Interface to X-13-ARIMA-SEATS, 2016. URL <https://cran.r-project.org/web/packages/seasonal/index.html>. [p29]

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6, 1978. [p17]

C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy. plotly: Create

Interactive Web Graphics via 'plotly.js', 2016. URL <http://cran.r-project.org/package=plotly>. [p4] D. F. Specht. A general regression neural network. IEEE Transactions on Neural Networks, 1991. [p10, 19] J.

Ulrich. TTR: Technical Trading Rules, 2016. URL <http://cran.r-project.org/package=TTR>. [p4]

D. Vanderkam, J. J. Allaire, J. Owen, D. Gromer, P. Shevtsov, and B. Thieurmel. dygraphs: Interface to 'Dygraphs' Interactive Time Series Charting Library, 2016. URL <https://cran.r-project.org/web/packages/dygraphs/index.html>. [p19]

H. Wickham and W. Chang. ggplot2: An Implementation of the Grammar of Graphics, 2015. URL <http://cran.r-project.org/package=ggplot2>. [p4]

A. Zeileis, G. Grothendieck, J. A. Ryan, and F. Andrews. zoo: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations), 2016. URL <http://cran.r-project.org/package=zoo>. [p4]