Project Report: "Be-FaVOR: Be Stars - Facilities in VO Research. Uma ferramenta para a determinação de parâmetros de estrelas Be"

Release 1.0

Artur Alegre, Alex C. Carciofi e Bruno C. Mota

November 02, 2016

1 Introdução **3**

2 Dados Observacionais **5**
2.1 Os dados do IUE 5
2.2 O *Web Service* 5

3 Ferramentas **7**
3.1 BeFaVOR-Web 7
3.2 TAKE-VIZIER-DATA 11

4 Outras Atividades e Perspectivas **13**
4.1 Atividades complementares 13
4.2 Tarefas futuras 13

5 Conclusões **15**

6 Anexos **17**
6.1 Anexo 1: Rotina BeFaVOr_web 17
6.2 Anexo 2: Rotina TAKE-VIZIER-DATA 29

7 Referências e INDEX **35**
7.1 Referências 35

Python Module Index **37**

Index **39**

O Observatório Astronômico Virtual (VO) é uma iniciativa internacional liderada pela *International Virtual Observatory Alliance* (IVOA), que visa integrar, através de ferramentas interoperacionais, diferentes bancos de dados e diferentes serviços no sentido de modelos, ferramentas de análise e outros.

Neste contexto, nosso trabalho visa disponibilizar uma ferramenta de análise *online* que proporcionará uma interface de integração entre o usuário e ferramentas desenvolvidas pelo grupo de pesquisa Beacon, coordenado pelo professor Dr. Alex C. Carciofi.

Esta ferramenta *online* irá permitir ao usuário estimar as propriedades fundamentais básicas de estrelas Be clássicas a partir de dados fotométricos. A base para esta ferramenta são os dados disponíveis em inúmeras bases de dados do VO, bem como os códigos HDUST de transferência radiativa e o código EMCEE de minimização de múltiplos parâmetros.

CHAPTER

ONE

INTRODUÇÃO

Apesar de terem sido descobertas há 150 anos por Secchi 1866, estrelas Be continuam sendo verdadeiros enigmas. Estrelas Be clássicas são estrelas que apresentam a maior taxa de rotação entre as estrelas da Sequência Principal (Rivinius, Carciofi & Martayan (2013)). Hoje, há consenso na comunidade de que a alta taxa de rotação está na base do surgimento dos efeitos peculiares apresentados por estas estrelas.

Um dos problemas centrais no estudo de estrelas do tipo Be é a obtenção de parâmetros fundamentais da estrela, tais como massa, idade, taxa de rotação e inclinação. Suas características particulares tornam essa tarefa difícil: estrelas Be são achatadas em seus polos, apresentam efeito de escurecimento gravitacional e, quando vistas de ângulos diferentes, apresentam distribuições espectrais de energia (SED) com diferentes características. A característica mais marcante, entretanto, é a presença de um disco circunstelar que altera de forma significativa o espectro estelar.

Buscando uma solução para este problema, o presente projeto visa disponibilizar à comunidade uma ferramenta no formato de um *web service*, isto é, um serviço de Observatório Virtual, ligado à infraestrutura do BRAVO - *Brazilian Virtual Observatory*. O método a ser utilizado por esta ferramenta baseia-se no confronto entre grades de modelos fotosféricos que levam em consideração a rotação e espectros da missão IUE, retornando dados confiáveis com estimativa de erro robusta de parâmetros estelares de uma dada lista de estrelas.

CHAPTER

TWO

DADOS OBSERVACIONAIS

2.1 Os dados do IUE

A missão *International Ultraviolet Explorer* (IUE) foi um projeto conjunto entre NASA, ESA e PPARC. Ainda hoje este projeto é entendido como um dos telescópios astronômicos mais produtivos de todos os tempos, ultrapassando as expectativas de seus objetivos originais, dentre estes, a obtenção de espectros de alta resolução de estrelas de todos os tipos espectrais para determinar suas características físicas e fazer repetidas observações de objetos com espectros variáveis.

Os arquivos de espectros do IUE podem ser acessados no sistema da ESA chamado IUE *Newly Extracted Spectra* (INES), cujo objetivo é fornecer à comunidade científica acesso aos espectros do IUE sem que se faça necessário um conhecimento técnico dos instrumentos. Para informações detalhadas sobre o sistema, veja http://sdc.cab.inta-csic.es/ines.

2.2 O *Web Service*

A ferramenta em desenvolvimento utilizará como *input* o nome de uma estrela para automaticamente baixar os dados do repositório *online* do INES e de outros serviços disponíveis e graficá-los de forma pré-derterminada. Desta maneira, não será necessário o usuário baixar os dados das estrelas que deseja estudar em seu computador, selecioná-los e, então, enviá-los de volta ao servidor para serem analisados; bastará apenas selecionar os espectros com os quais deseja trabalhar.

Para construir esta ferramenta, o projeto foi dividido em duas etapas:

1. Elaboração de uma rotina capaz de ler os dados observacionais diretamente através do servidor.

   Esta rotina é a responsável por fazer o *download* automático dos dados buscando no servidor pelo nome da estrela.

2. Implementação da rotina Python de determinação dos parâmetros estelares.

   Esta parte inclui a criação de uma interface que permita ao usuário executar as rotinas de interesse dentro de um ambiente de VO.

Este serviço será hospedado na *homepage* do grupo Beacon, ao lado de outro projeto que visamos integrar ao BRAVO, o BEATLAS (Faes et al. 2017 in prep.), ferramenta que colocará à disposição da comunidade cerca de 70.000 espectros sintéticos de estrelas Be.

CHAPTER

THREE

FERRAMENTAS

Nesta seção, são descritas as rotinas Python criadas para leitura dos dados através do servidor: BeFaVOR-Web e TAKE-VIZIER-DATA. Esta última trata-se de uma rotina complementar para habilitar a primeira a buscar por múltiplas estrelas de uma vez.

3.1 BeFaVOR-Web

A rotina BeFaVOR-Web usa como *input* tanto o nome de uma única estrela quanto uma lista de estrelas para realizar uma busca por arquivos de espectros correspondentes no *database* do sistema INES, salvando-os na pasta definida. Em seguida, realiza-se, para a mesma estrela, uma busca no SIMBAD, através do qual obtemos os parâmetros: paralaxe, vsini e suas respectivas incertezas, além da referência bibliográfica para vsini. Também é verificado se a busca na plataforma IRSA-*Galactic DUST Reddening & Extinction* retorna algum valor de avermelhamento E(B-V) para aquela estrela. Caso positivo, o valor é salvo juntamente com os outros parâmetros em uma tabela na pasta definida.

3.1.1 Instalação

Para utilizar esta rotina em sua atual versão é preciso ter instalados os seguintes pacotes:

1. numpy
2. bs4
3. datetime
4. requests
5. selenium
6. astroquery
7. pyhdust
8. matplotlib
9. urllib
10. re
11. random
12. tafile

Os pacotes numerados de 1 a 8 podem ser instalados utilizando o gerenciador de pacotes **pip**, enquanto aqueles de 9 a 12 podem ser instalados pelo comando **apt-get install python3-pacote**.

3.1.2 Executando a BeFaVOR-Web

Para rodar a rotina o usuário deve primeiro definir o *path* onde esta encontra-se instalada na tag **commum_folder**, bem como o *path* onde as tabelas serão salvas em **folder_tables**. Recomenda-se utilizar a versão mais atual disponível do Python.

Utilizando como exemplo o IPython 3.0, o código pode ser rodado seguindo os seguintes passos:

1. Abra o terminal e digite o comando: ipython3
2. Vá ao diretório em que a rotina encontra-se instalada;
3. No terminal, digite o comando: run BeFaV0r_web.py
4. A rotina perguntará se o usuário deseja buscar apenas um ou vários alvos. Digite '1' para buscar um único alvo ou 'more' para buscar múltiplos alvos;
5. Caso tenha digitado '1', a rotina perguntará o nome do alvo. Digite o nome da estrela desejada;
6. Caso tenha digitado 'more' a rotina executará a rotina complementar TAKE_VIZIER_DATA que baixará uma série de alvos de acordo com um catálogo pré-definido. Para mais detalhes, veja a descrição da rotina TAKE_VIZIER_DATA na sub-seção seguinte.

Se o usuário tiver buscado por um único alvo, além dos espectros do IUE salvos, a rotina criará um documento de texto com a seguinte estrutura de **linhas**:

1. Nome da estrela
2. Paralaxe
3. Incerteza da paralaxe
4. vsini
5. Incerteza do vsini
6. "Bump"
7. Referência bibliográfica do vsini
8. E(B-V)

"Bump" refere-se à presença de um espectro IUE que cobre a região conhecida como *bump* 2200 Angstron que é utiliza na determinação do valor de E(B-V). Caso retorne "*True*", existe este espectro e, caso retorne "*False*", significa que o espectro não cobre esta faixa espectral e, portanto, o valor é retirado do IRSA.

Se o usuário tiver buscado por múltiplos alvos, além dos espectros do IUE salvos, a rotina criará um documento de texto com uma estrutura de **colunas** pré-determinada pelo usuário. Por exemplo, caso o usuário selecione estrelas do *Bright Star Catalogue*, é possível gerar a seguinte estrutura:

1. Nome da estrela
2. Índice B-V
3. Índice U-B
4. Índice R-I
5. vsini
6. Incerteza B-V
7. Incerteza U-B
8. Incerteza vsini
9. Incerteza R-I

10. Data de observação
11. Tipo espectral

3.1.3 Funções da rotina BeFaVOR-Web

Segue a descrição das funções contidas na rotina BeFaVOR-Web, organizadas em ordem alfabética:

BeFaVOr_web.**create_list_files**(*list_name*, *folder*, *folder_table*)
Creates a list of the files inside a given folder.

Parameters

- **list_name** – list's name (string)
- **folder** – files' folder (string)

Returns creates a txt file, with the files' paths

BeFaVOr_web.**create_txt_file**(*data_list*, *file_name*)
Create a txt file.

Parameters

- **data_list** – list of data to be saved (array)
- **file_name** – txt file's name (string)

Returns txt file

BeFaVOr_web.**find_regular_expression**(*url*, *typ*, *atr*, *expr*)
Search for a specified expression inside a page code and lists all its ocurrences.

Parameters

- **url** – page url (string)
- **typ** – tag (string)
- **atr** – attribute (string)
- **expr** – expression (string)

Returns list of occurences

BeFaVOr_web.**getTitle**(*url*)
Gets the title of a certain webpage.

Parameters **url** – page url (string)

Returns page title

BeFaVOr_web.**get_attribute**(*url*, *atr*, *typ*)
Lists all text in a webpage that satisfacts a specified attribute.

Parameters

- **url** – page url (string)
- **atr** – attribute (string)
- **typ** – tag (string)

Returns text containing specified attribute

BeFaVOr_web.**iue_submission**(*star_name*)
Search in the IUE database for a certain star name.

Parameters **star_name** – name of the star (string)

Returns request of star name in IUE page

BeFaVOr_web.**plot_gal**(*ra_val*, *dec_val*, *folder_fig*)
Plot in "Galatic Coordinates" (i.e., Mollweide projection).

Parameters

- **ra_val** – right ascencion in RADIANS (float)
- **dec_val** – declination in RADIANS (float)
- **folder_fig** – name of the folder for the figure (string)

Returns saved images

BeFaVOr_web.**read_simbad_coodr**(*star_name*)
Query SIMBAD for the coordinates of a given star.

Parameters **star_name** – star's name (string)

Returns right ascencion and declination coordinates

BeFaVOr_web.**read_simbad_data**(*star_name*)
Query SIMBAD for a given star parallax, vsini, bump and references.

Parameters **star_name** – star's name (string)

Returns txt file with star's parallax, vsini, bump, the respective

errors and references for vsini and E(B-V)

BeFaVOr_web.**read_txt**(*list_name*, *folder*)
Read a given list of star names.

Parameters

- **list_name** – name o txt file containing the list (string)
- **folder** – list's folder (string)

Returns column of the list read

BeFaVOr_web.**retrieve_ebmv_value**(*star_name*)
Search the INES website for a specified star's E(B-V) value.

Parameters **star_name** – stars's name (string)

Returns E(B-V) value

BeFaVOr_web.**selecting_data**(*star_name*, *commum_folder*)
Search the INES website for a specified star.

Parameters

- **star_name** – name of the star (string)
- **commum_folder** – name of the folder where the routine is located

Returns request of the star name in INES page

BeFaVOr_web.**show_page_code**(*url*)
Shows the page code of a certain webpage.

Parameters **url** – page url (string)

Returns page code

BeFaVOr_web.**there_is_a_title**(*url*)
Prints the title of a certain webpage.

Parameters **url** – page url (string)

Returns page title

BeFaVOr_web.**untar**(*fname*)
Decompact a tar file.

Parameters **fname** – name o file to be decompacted (string)

Returns decompacted file

BeFaVOr_web.**unzip**(*zip_file*, *outdir*)
Unzip a given file into the specified output directory.

Parameters

- **zip_file** – name of file to be unziped (string)
- **outdir** – directory of the file (string)

Returns unziped file

3.2 TAKE-VIZIER-DATA

A rotina TAKE-VIZIER-DATA realiza uma busca por catálogos no VizieR. Em seguida, dentro de cada catálogo selecionado, é feita a seleção dos objetos de interesse através de um processo de filtragem por tipo espectral, classe de luminosidade e existência ou não de dados da missão IUE. Deste ponto em diante, a rotina segue os seguintes passos:

1. As estrelas selecionadas no VizieR são buscadas no arquivo do INES;
2. Os espectros daquelas encontradas são salvos no diretório definido;
3. Para cada estrela encontrada no INES, é criado um documento de texto onde são salvos os parâmetros retirados do SIMBAD e (opcional) os parâmetros obtidos dos catálogos lidos por meio do VIZIER.

Detalhes sobre as funções contidas nesta rotina podem ser vistos no anexo do capítulo 6.

CHAPTER

FOUR

OUTRAS ATIVIDADES E PERSPECTIVAS

4.1 Atividades complementares

Ao longo do período de duração desta bolsa, além das tarefas diretamente relacionadas a este projeto, também foram realizadas atividades importantes para o desenvolvimento acadêmico, dentre elas destacamos:

1. Participação de reuniões semanais do grupo Beacon, onde são discutidos resultados recentes das pesquisas de seus membros, bem como artigos científicos relevantes para nossa linha de pesquisa;
2. Apresentação de resultados parciais deste projeto na reunião de grupo;
3. Participação de observações astronômicas tanto remotas quanto presenciais no Observatório Pico dos Dias para obtenção de dados polarimétricos e espectroscópicos.

4.2 Tarefas futuras

1. Incorporação das rotinas de leitura de dados em servidores desenvolvidas neste projeto com a rotina Python de determinação de parâmetros estelares que constitui parte do projeto de doutorado do aluno Bruno C. Mota.
2. Desenvolvimento e integração, na *homepage* do grupo Beacon, da interface que permitirá ao usuário executar as rotinas Python criadas ao longo ou anteriormente a este projeto em um ambiente de Observatório Virtual.

CHAPTER

FIVE

CONCLUSÕES

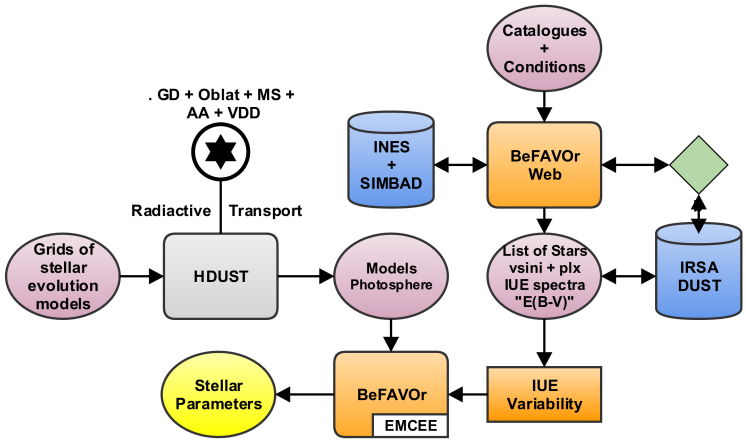

Figure 5.1: Fluxograma simplificado do método desenvolvido.

Na figura _fig1, mostramos uma representação simplificada do método que vem sendo desenvolvido. Neste projeto de iniciação, devenvolvemos o bloco *BeFaVOR web* representado na figura. Em resumo, o usuário necessita apenas selecionar catálogos de interesse, bem como os filtros adequados. A partir deste ponto, a rotina verifica se as estrelas selecionadas possuem dados IUE; caso positivo, dados de paralaxe e vsini são lidos e salvos automaticamente do SIMBAD, juntamente com os espectros do IUE (salvos da plataforma INES). Uma implementação adicional é a verificação da existência de uma região espectral que pode ser utilizada para se determinar o avermelhamento devido ao meio interestelar. Caso ela não exista, a rotina lê e salva uma estimativa do *web service* IRSA. Por fim, o usuário obtêm uma lista de estrelas, com seus respectivos dados de paralaxe, vsini, E(B-V) e incertezas. Estes dados são utilizados na rotina BeFaVOR de determinação dos parâmetros estelares, que está sendo desenvolvida pelo doutorando Bruno C. Mota.

CHAPTER

SIX

ANEXOS

6.1 Anexo 1: Rotina BeFaVOr_web

```
# 1) Esta rotina deve ser rodada em ipython3

# ================================================================================
# Importing modules
import numpy as np
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup
import sys
import re
import datetime
import random
import time
import requests
import math
from selenium import webdriver
from selenium.webdriver.support.ui import Select
import tarfile
from astroquery.simbad import Simbad
import csv
import os
from glob import glob
import pyhdust.phc as phc
import matplotlib.pyplot as plt
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC


# ================================================================================
def show_page_code(url):
    '''
    Shows the page code of a certain webpage.

    :param url: page url (string)
    :return: page code
    '''

    html = urlopen(url)
    bsObj = BeautifulSoup(html.read())
```

```
    return bsObj


# ================================================================================
def getTitle(url):
    '''
    Gets the title of a certain webpage.

    :param url: page url (string)
    :return: page title
    '''

    try:
        html = urlopen(url)
    except HTTPError as e:
        print(e)
        return None
    try:
        bsObj = BeautifulSoup(html.read())
        title = bsObj.body.h1
    except AttributeError as e:
        return None
    return title


# ================================================================================
def there_is_a_title(url):
    '''
    Prints the title of a certain webpage.

    :param url: page url (string)
    :return: page title
    '''

    title = getTitle(url)
    if title is None:
        print("Title could not be found")
    else:
        print(title)
    return title


# ================================================================================
def example_try():

    while True:
        try:
            x = int(raw_input("Please enter a number: "))
            break
        except ValueError:
            print("Oops!  That was no valid number.  Try again...")
    return


# ================================================================================
def get_attribute(url, atr, typ):
    '''
    Lists all text in a webpage that satisfacts a specified attribute.
```

```
    :param url: page url (string)
    :param atr: attribute (string)
    :param typ: tag (string)
    :return: text containing specified attribute
    '''

    bsOBJ = show_page_code(url)
    namelist = bsOBJ.findAll(typ, {"class": atr})
    names = []
    for name in namelist:
        nome = name.get_text()
        names.append(nome)

    return names


# ================================================================================
def find_regular_expression(url, typ, atr, expr):
    '''
    Search for a specified expression inside a page code and lists all
    its ocurrences.

    :param url: page url (string)
    :param typ: tag (string)
    :param atr: attribute (string)
    :param expr: expression (string)
    :return: list of occurences
    '''

    html = urlopen(url)
    bsObj = BeautifulSoup(html)
    array = bsObj.findAll(typ, {atr: re.compile(expr)})
    for lista in array:
        print(lista["src"])
    return lista


# ================================================================================
random.seed(datetime.datetime.now())


def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org" + articleUrl)
    bsObj = BeautifulSoup(html)
    return bsObj.find("div", {"id": "bodyContent"}).\
        findAll("a", href=re.compile("^(/wiki/)((?!:).)*$"))


# ================================================================================
def get_Link(articleUrl):
    links = getLinks("/wiki/Kevin_Bacon")
    while len(links) > 0:
        newArticle = links[random.randint(0, len(links) - 1)].attrs["href"]
        print(newArticle)
        links = getLinks(newArticle)


# ================================================================================
```

```
def file_submission():
    files = {'uploadFile': open('../files/Python-logo.png', 'rb')}
    r = requests.post("http://pythonscraping.com/pages/processing2.php",
                      files=files)
    print(r.text)
    return


# ========================================================================
def simple_form():
    params = {'firstname': 'Ryan', 'lastname': 'Mitchell'}
    r = requests.post("http://pythonscraping.com/files/processing.php",
                      data=params)
    print(r.text)
    return


# ========================================================================
def file_submission2():
    files = {'uploadFile': open('../files/python.png', 'rb')}
    r = requests.post("http://pythonscraping.com/pages/processing2.php",
                      files=files)
    print(r.text)

    return


# ========================================================================
def iue_submission(star_name):
    '''
    Search in the IUE database for a certain star name.

    :param star_name: name of the star (string)
    :return: request of star name in IUE page
    '''

    ines_site = "http://sdc.cab.inta-csic.es/cgi-ines/IUEdbsMY"
    params = {'object': star_name}
    r = requests.post(ines_site, data=params)
    print(r.text)

    return r


# ========================================================================
def read_txt(list_name, folder):
    '''
    Read a given list of star names.

    :param list_name: name o txt file containing the list (string)
    :param folder: list's folder (string)
    :return: column of the list read
    '''

    list_name = folder + list_name

    cols = np.genfromtxt(list_name, unpack=True, comments='#',
                         delimiter='\t')
```

```
    return cols


# ============================================================================
def untar(fname):
    '''
    Decompact a tar file.

    :param fname: name o file to be decompacted (string)
    :return: decompacted file
    '''

    if (fname.endswith("tar.gz")):
        tar = tarfile.open(fname)
        tar.extractall()
        tar.close()
        print("Extracted in Current Directory")
    else:
        print("Not a tar.gz file: '%s '" % sys.argv[0])


# ============================================================================
def create_list_files(list_name, folder, folder_table):
    '''
    Creates a list of the files inside a given folder.

    :param list_name: list's name (string)
    :param folder: files' folder (string)
    :return: creates a txt file, with the files' paths
    '''

    a = open(folder_table + list_name + ".txt", "w")
    for path, subdirs, files in os.walk(folder):
        for filename in files:
            f = os.path.join(path, filename)
            a.write(str(f) + os.linesep)
    return


# ============================================================================
def create_txt_file(data_list, file_name):
    '''
    Create a txt file.

    :param data_list: list of data to be saved (array)
    :param file_name: txt file's name (string)
    :return: txt file
    '''

    with open(file_name, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        writer.writerows(zip(data_list))
    return


# ============================================================================
def read_simbad_data(star_name):
    '''
```

```
    Query SIMBAD for a given star parallax, vsini, bump and references.

    :param star_name: star's name (string)
    :return: txt file with star's parallax, vsini, bump, the respective
    errors and references for vsini and E(B-V)
    '''

    customSimbad = Simbad()
    # customSimbad.list_votable_fields() #  to list all avaiable fields
    customSimbad.get_votable_fields()
    customSimbad.add_votable_fields('plx', 'plx_error', 'rot')
    customSimbad.get_votable_fields()
    result_table = customSimbad.query_object(star_name)
    # star = result_table['MAIN_ID']
    star = np.copy(star_name)
    plx = result_table['PLX_VALUE'][0]
    plx_error = result_table['PLX_ERROR'][0]
    vsini = result_table['ROT_Vsini'][0]
    vsini_err = result_table['ROT_err'].item()
    rot_bibcode = result_table['ROT_bibcode']
    bump = True
    ebmv_ref = 0.0

    return star.item(), plx, plx_error, vsini, vsini_err, bump,\
        rot_bibcode.item(), ebmv_ref


# ============================================================================
def read_simbad_coodr(star_name):
    '''
    Query SIMBAD for the coordinates of a given star.

    :param star_name: star's name (string)
    :return: right ascencion and declination coordinates
    '''

    customSimbad = Simbad()
    customSimbad.get_votable_fields()
    result_table = customSimbad.query_object(star_name)

    ra = result_table['RA'][0]
    dec = result_table['DEC'][0]

    return ra, dec


# ============================================================================
def unzip(zip_file, outdir):
    '''
    Unzip a given file into the specified output directory.

    :param zip_file: name of file to be unziped (string)
    :param outdir: directory of the file (string)
    :return: unziped file
    '''

    import zipfile
    zf = zipfile.ZipFile(zip_file, "r")
```

```
    zf.extractall(outdir)
    return


# ============================================================================
def plot_gal(ra_val, dec_val, folder_fig):
    """
    Plot in "Galatic Coordinates" (i.e., Mollweide projection).

    :param ra_val: right ascencion in RADIANS (float)
    :param dec_val: declination in RADIANS (float)
    :param folder_fig: name of the folder for the figure (string)
    :return: saved images
    """

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="mollweide")
    ax.set_xticklabels(['14h', '16h', '18h', '20h', '22h', '0h', '2h', '4h',
                        '6h', '8h', '10h'])

    for i in range(len(ra_val)):
        dec = dec_val[i].replace(' ', ':')
        ra = ra_val[i].replace(' ', ':')

        # list of floats (degrees fraction)
        dec = [phc.dec2degf(dec)]
        ra = [phc.ra2degf(ra)]

        # arrays of floats (radians)
        dec = np.array(dec) * np.pi / 180
        ra = np.array(ra) * np.pi / 180

        ax.scatter(ra, dec)
    plt.savefig(folder_fig + 'galatic_distribution.png')

    return


# ============================================================================
def selecting_data(star_name, commum_folder):
    '''
    Search the INES website for a specified star.

    :param star_name: name of the star (string)
    :param commum_folder: name of the folder where the routine is located
    :return: request of the star name in INES page
    '''

    from pyvirtualdisplay import Display
    display = Display(visible=0, size=(800, 600))
    display.start()

    # now Chrome will run in a virtual display.
    # you will not see the browser.

    # Creating the path
    a = star_name.split()
```

```
    # short_star_name = a[0][0] + a[1][0:3]
    short_star_name = a[0] + a[1]

    # Starting the searching
    if os.path.isdir(commum_folder + 'iue/' + short_star_name) is False:
        os.mkdir(commum_folder + 'iue/' + short_star_name)

        folder_data = commum_folder + 'iue/' + short_star_name

        # Define global Chrome properties
        options = webdriver.ChromeOptions()
        prefs = {"download.default_directory": folder_data}
        options.add_experimental_option("prefs", prefs)

        browser = webdriver.Chrome(chrome_options=options)
        # browser = webdriver.Firefox(firefox_profile=fp)

        # Define web source
        ines_site = "http://sdc.cab.inta-csic.es/cgi-ines/IUEdbsMY"

        # Openning it
        browser.get(ines_site)
        # browser.maximize_window()
        # time.sleep(3)

        # Selecting all data
        mySelect = Select(browser.find_element_by_name("limit"))
        mySelect.select_by_value("all")
        # time.sleep(3)

        # Selecting some stars
        browser.find_element_by_name("object").send_keys(star_name)
        browser.find_element_by_name(".submit").click()
        # time.sleep(3)

        # Taking the data
        browser.find_element_by_name("markRebin").click()
        browser.find_element_by_name(".submitNH").click()
        time.sleep(20)
        # browser.close()

        # Unzip files
        outdir = os.getcwd()
        # print(short_star_name)
        # new_path = outdir + '/iue/' + short_star_name + '/'
        os.chdir(folder_data)
        file_list = glob('*')
        if len(file_list) != 0:
            # print(file_list)
            fname = str(file_list[0])
            # print(fname)
            tar = tarfile.open(fname, "r:gz")
            tar.extractall()
            tar.close()
            os.system('rm *.gz')
        os.chdir(outdir)
        browser.close()
```

```
    return


# ============================================================================
def retrieve_ebmv_value(star_name):
    '''
    Search the INES website for a specified star's E(B-V) value.

    :param star_name: stars's name (string)
    :return: E(B-V) value
    '''

    from pyvirtualdisplay import Display
    display = Display(visible=0, size=(800, 600))
    display.start()

    # Define global Chrome properties
    browser = webdriver.Chrome()

    # Define web source
    irsa_site = "http://irsa.ipac.caltech.edu/applications/DUST/"

    # Openning it
    browser.get(irsa_site)
    # wait = WebDriverWait(browser, 180)
    # wait.until(EC.title_contains("title"))

    # Selecting some stars
    browser.find_element_by_name("locstr").send_keys(star_name)
    browser.find_element_by_class_name("tdsubmit").click()

    time.sleep(30)
    ebmv = browser.find_element_by_class_name("tdwhiteleft")
    ebmv = float(ebmv.text)
    browser.close()

    return ebmv

    return ebmv


# ============================================================================
def main():

    num_spa = 75
    print(num_spa * '=')
    print('\nBeFaVOr_Web\n')
    print(num_spa * '=')
# ----------------------------------------------------------------------------

    # Defining folders
    user = input('Who is using? (bmota or artur): ')
    commum_folder = '/home/' + user + '/Dropbox/Artur/BeFaVOr_web/'
    commum_folder_2 = '/home/' + user + '/Dropbox/Artur/BeFaVOr_web/'

    folder_tables = commum_folder + 'tables/'
    folder_tables_2 = commum_folder_2 + 'emcee/' + 'tables/'
```

```
    folder_figures = commum_folder + 'figures/'

    table_final = folder_tables + 'list.txt'
    table_final_2 = folder_tables_2 + 'list_final.txt'
# ----------------------------------------------------------------------------

    # Saving the input for the routine Befavour.py
    if os.path.isfile(table_final_2) is True:
        os.system('rm ' + table_final_2)

    if os.path.isfile(folder_figures) is False:
        os.system('rm -r ' + folder_figures)

    if os.path.isfile(folder_figures) is False:
        os.system('mkdir ' + folder_figures)

    if os.path.isfile(table_final) is True:
        os.system('rm ' + table_final)

    os.system('rm -r' + folder_tables)
    if os.path.isdir(commum_folder + 'tables/') is False:
        os.mkdir(commum_folder + 'tables/')
# ----------------------------------------------------------------------------

    # Would you like to run one or a list of stars?
    option = input('\nRun one or more stars: (1 or more) ')

    if option == '1':
        # Saving data from the INES database
        star = input('\nPlease, put the star name: ')
        print('\nSaving data from INES database...')
        selecting_data(star_name=star, commum_folder=commum_folder)

        # Saving SIMBAD stellar data to a table
        print('\nSaving data (plx, vsini) from SIMBAD database...')
        table_file = commum_folder + 'tables/' + star + '.txt'
        folder_tables = commum_folder + 'tables/'
        val = read_simbad_data(star_name=star)

        # Check if there are bump files
        folder_star = commum_folder + 'iue/' + star
        bump = glob(folder_star + 'L*')

        if len(bump) is 0:
            start_time = time.time()
            val = list(val)
            ebmv_bump = retrieve_ebmv_value(star_name=star)
            print(ebmv_bump)
            val[7] = ebmv_bump
            val[5] = False
            val = tuple(val)
            print("--- %s seconds ---" % (time.time() - start_time))

        # Saving the table
        create_txt_file(data_list=val, file_name=table_file)
# ----------------------------------------------------------------------------

    if option == 'more':
```

```
        cols = read_txt(list_name='selected_bn_stars.txt',
                        folder=commum_folder + 'tables_vizier/')

        cols_2 = read_txt(list_name='selected_bn_stars_compl.txt',
                          folder=commum_folder + 'tables_vizier/')

        cols_3 = read_txt(list_name='selected_be_stars.txt',
                          folder=commum_folder + 'tables_vizier/')

        cols_4 = read_txt(list_name='selected_be_bsc_stars.txt',
                          folder=commum_folder + 'tables_vizier/')

        # cols_5 = read_txt(list_name='selected_be_bsc_stars_compl.txt',
        #                   folder=commum_folder + 'tables_vizier/')

        stars = np.concatenate((cols[0], cols_2[0], cols_3[0],
                                cols_4[0]), axis=0)

        # stars = cols_3[0]

# ----------------------------------------------------------------------------

        for i in range(len(stars)):
            star = str(stars[i])
            star = "HD " + star[:-2]
            a = star.split()
            star_2 = a[0] + a[1]
            # Saving data from the INES database
            print(num_spa * '=')
            print('\nStar: %s' % star)
            print('\nSaving data from INES database... %d of %d' %
                  (i + 1, len(stars)))

            selecting_data(star_name=star, commum_folder=commum_folder)

            # Saving SIMBAD stellar data to a table
            print('\nSaving data from SIMBAD database... star: %s\n'
                  % (star))
            table_file = commum_folder + 'tables/' + star + '.txt'
            val = read_simbad_data(star_name=star)

            # Check if there are bump files
            folder_star = commum_folder + 'iue/' + star_2 + '/'

            bump = glob(folder_star + 'L*')
            # print(star_2)
            # print(folder_star)
            # print(bump)
            # print(len(bump))
            if len(bump) is 0:
                start_time = time.time()
                val = list(val)
                ebmv_bump = retrieve_ebmv_value(star_name=star)
                print(ebmv_bump)
                val[7] = ebmv_bump
                val[5] = False
                val = tuple(val)
                print("--- %s seconds ---" % (time.time() - start_time))
```

```
                print(val)
            # nan and "--" Filters
            if math.isfinite(val[1]) is True and math.isfinite(val[2]) is True\
               and math.isfinite(val[3]) is True:

                if math.isfinite(val[3]) is True and\
                   math.isfinite(val[4]) is False:
                    val = list(val)
                    val[4] = 0.0
                    val = tuple(val)
                    create_txt_file(data_list=val, file_name=table_file)
                else:
                    create_txt_file(data_list=val, file_name=table_file)
            else:
                print('This Star was excluded!')
# ----------------------------------------------------------------------------

        # Plotting galatic distribution
        ra_val_arr = []
        dec_val_arr = []
        for i in range(len(stars)):
            star = str(stars[i])
            star = "HD " + star[:-2]
            ra, dec = read_simbad_coodr(star_name=star)
            ra_val_arr.append(ra)
            dec_val_arr.append(dec)

        plot_gal(ra_val=ra_val_arr, dec_val=dec_val_arr,
                 folder_fig=folder_figures)

# ----------------------------------------------------------------------------

    # Creating list of files
    create_list_files(list_name='list', folder=folder_tables,
                      folder_table=folder_tables)

    table_final = folder_tables + 'list.txt'
    table_final_2 = folder_tables_2 + 'list_final.txt'

    files = open(table_final)
    files = files.readlines()
    final_table = open(table_final_2, "a+")

    for i in range(len(files)):
        files_2 = open(files[i][:-1])

        lines = files_2.readlines()

        if len(lines) == 8:
            star = lines[0][:-1]
            a = star.split()

            if len(a) >= 2:
                short_star_name = a[0] + a[1]
            else:
                short_star_name = np.copy(a)

            final_table.writelines(('%s\t%s\t%s\t%s\t%s\t%s\t%s\n')
```

```
                                        % (short_star_name, lines[1][:-1],
                                           lines[2][:-1], lines[3][:-1],
                                           lines[4][:-1], lines[7][:-1],
                                           lines[5][:-1]))
    final_table.close()
    print(num_spa * '=')
    print('\nFinished\n')
    return


# ============================================================================
if __name__ == '__main__':
    main()
```

6.2 Anexo 2: Rotina TAKE-VIZIER-DATA

```
# ============================================================================
# !/usr/bin/env python
# -*- coding:utf-8 -*-

# Created by B. Mota 2016-02-16 to present...

# import packages

import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.font_manager as fm
import numpy as np
import pyhdust.phc as phc
from astroquery.vizier import Vizier
from astroquery.simbad import Simbad
import csv
import os
# import pyraf
mpl.rcParams.update({'font.size': 18})
mpl.rcParams['lines.linewidth'] = 2
font = fm.FontProperties(size=17)
mpl.rc('xtick', labelsize=17)
mpl.rc('ytick', labelsize=17)
fontsize_label = 18  # 'x-large'

__version__ = "0.0.1"
__author__ = "Bruno Mota"

# ============================================================================
# Parameters that must be defined
user = 'bruno'
num_spa = 75
commum_folder = '/home/' + user + '/Dropbox/Artur/BeFaVOr_web/' +\
    'tables_vizier/'
folder_fig = '/home/' + user + '/Dropbox/Artur/BeFaVOr_web/figures/'

if os.path.isdir(folder_fig) is False:
    os.mkdir(folder_fig)

print(num_spa * '=')
```

```
print('\nTake_VIZIER_data\n')
print(num_spa * '=')


# ============================================================================
def create_txt_file(a, b, c, d, e, f, g, h, i, j, l, file_name):
    '''
    Create a txt file.

    :param a: table's column (array)
    :param b: table's column (array)
    :param c: table's column (array)
    :param d: table's column (array)
    :param e: table's column (array)
    :param f: table's column (array)
    :param g: table's column (array)
    :param h: table's column (array)
    :param i: table's column (array)
    :param j: table's column (array)
    :param l: table's column (array)
    :param file_name: file's name (string)
    :return: txt file
    '''

    with open(file_name, 'w') as k:
        # file.write(('%s\t & \t %s \t & \t %s \t & \t %s \t & \t  %s \t &' +
        #                ' \t  %s \t & \t  %s   \n')
        writer = csv.writer(k, delimiter='\t')
        writer.writerows(zip(a, b, c, d, e, f, g, h, i, j, l))

    return


# ============================================================================
def create_txt_file_compl(a, b, c, d, e, f, g, h, file_name):
    '''
    Create a txt file.

    :param a: table's column (array)
    :param b: table's column (array)
    :param c: table's column (array)
    :param d: table's column (array)
    :param e: table's column (array)
    :param f: table's column (array)
    :param g: table's column (array)
    :param h: table's column (array)
    :param file_name: file's name (string)
    :return: txt file
    '''

    with open(file_name, 'w') as k:
        writer = csv.writer(k, delimiter='\t')
        writer.writerows(zip(a, b, c, d, e, f, g, h))

    return


# ============================================================================
```

```
def main():

    Vizier.ROW_LIMIT = -1  # VIZIER whole catalog
    cat = ['V/50', 'V/36B']
    catalogs = Vizier.get_catalogs(cat)
    catalog = catalogs[0]
    catalog_compl = catalogs[2]

    # Operating with the data
    data = catalog.as_array()
    data_compl = catalog_compl.as_array()

    # Print available data
    data.dtype
    data_compl.dtype

    # Filtering the SpType
    sptype = list(data['SpType'].data)
    sptype_compl = list(data_compl['SpType'].data)
    # indexes = np.where(conc_flux[0] > 0)

    indexes = []
    for i in range(len(sptype)):
        sptyp = sptype[i].decode('UTF-8')
        if len(sptyp) != 0:
            if sptyp[0] == 'B':
                if ('e' in sptyp) is False:
                    if ('IV' in sptyp) is False:
                        if ('IIII' in sptyp) is False:
                            if ('Hg' in sptyp) is False:
                                if ('Mn' in sptyp) is False:
                                    if ('n' in sptyp) is True:
                                        indexes.append(i)

    indexes_compl = []
    for i in range(len(sptype_compl)):
        sptyp_compl = sptype_compl[i].decode('UTF-8')
        if len(sptyp_compl) != 0:
            if sptyp_compl[0] == 'B':
                if ('e' in sptyp_compl) is False:
                    if ('IV' in sptyp_compl) is False:
                        if ('IIII' in sptyp_compl) is False:
                            if ('Hg' in sptyp_compl) is False:
                                if ('Mn' in sptyp_compl) is False:
                                    if ('n' in sptyp_compl) is True:
                                        if ('n' in sptyp_compl) is True:
                                            indexes_compl.append(i)

# ============================================================================
    # Selecting the data with the B stars
    selected_data = data[indexes]
    sptyp_selected = list(selected_data['SpType'])
    name_selected = selected_data['Name']
    hd_selected = selected_data['HD']
    plx = selected_data['Parallax']
    bmv = selected_data['B-V']
    err_bmv = selected_data['u_B-V']
    umb = selected_data['U-B']
```

```
    err_umb = selected_data['u_U-B']
    rmi = selected_data['R-I']
    vsini = selected_data['RotVel']
    err_vsini = selected_data['u_RotVel']
    companions = selected_data['MultCnt']

    selected_data_compl = data_compl[indexes_compl]
    sptyp_selected_compl = list(selected_data_compl['SpType'])
    hd_selected_compl = selected_data_compl['HD']
    plx_compl = selected_data_compl['Plx']
    bmv_compl = selected_data_compl['B-V']
    umb_compl = selected_data_compl['U-B']
    rmi_compl = selected_data_compl['R-I']
    vsini_compl = selected_data_compl['vsini']
    err_vsini_compl = selected_data_compl['u_vsini']


# ============================================================================
    # Checking if there are IUE data
    customSimbad = Simbad()
    customSimbad.TIMEOUT = 2000  # sets the timeout to 2000s

    # see which fields are currently set
    customSimbad.get_votable_fields()

    # To set other fields
    customSimbad.add_votable_fields('measurements')


# ============================================================================
    # Selecting the stars with IUE data
    data = data[indexes]
    obs_iue_date = []
    stars = []
    indexes = []
    print(num_spa * '=')
    print('\nselected stars: %d\n' % len(hd_selected))
    print(num_spa * '=')
    for i in range(len(hd_selected)):
        try:
            star = "HD " + str(hd_selected[i])
            result_table = customSimbad.query_object(star)
            obs_date = result_table['IUE_ObsDate']
            if len(obs_date.item()) != 0:
                print(num_spa * '-')
                print('\n' + star)
                print('%0.2f perc. concluded' % (100 * i / len(hd_selected)))
                print(obs_date)
                obs_iue_date.append(obs_date.item())
                stars.append(star)
                indexes.append(i)
        except:
            pass

    data_compl = data_compl[indexes_compl]
    obs_iue_date_compl = []
    stars_compl = []
    indexes_compl = []
```

```
    print('selected stars compl: %d' % len(hd_selected_compl))
    for i in range(len(hd_selected_compl)):
        try:
            star = "HD " + str(hd_selected_compl[i])
            result_table = customSimbad.query_object(star)
            obs_date = result_table['IUE_ObsDate']
            if len(obs_date.item()) != 0:
                print(num_spa * '-')
                print('\n' + star)
                print('%0.2f perc. concluded' % (100 * i / len(hd_selected)))
                print(obs_date)
                obs_iue_date_compl.append(obs_date.item())
                stars_compl.append(star)
                indexes_compl.append(i)
        except:
            pass

# ============================================================================
    # Selecting the data with the B stars in IUE database

    selected_data = data[indexes]
    sptyp_selected = list(selected_data['SpType'])
    name_selected = selected_data['Name']
    hd_selected = selected_data['HD']
    plx = selected_data['Parallax']
    bmv = selected_data['B-V']
    err_bmv = selected_data['u_B-V']
    umb = selected_data['U-B']
    err_umb = selected_data['u_U-B']
    rmi = selected_data['R-I']
    vsini = selected_data['RotVel']
    err_vsini = selected_data['u_RotVel']
    companions = selected_data['MultCnt']

    selected_data_compl = data_compl[indexes_compl]
    sptyp_selected_compl = list(selected_data_compl['SpType'])
    hd_selected_compl = selected_data_compl['HD']
    plx_compl = selected_data_compl['Plx']
    bmv_compl = selected_data_compl['B-V']
    umb_compl = selected_data_compl['U-B']
    rmi_compl = selected_data_compl['R-I']
    vsini_compl = selected_data_compl['vsini']
    err_vsini_compl = selected_data_compl['u_vsini']


# ============================================================================
    # Plotting correlations

    # Plot B-V vs U-B
    plt.clf()
    plt.scatter(bmv, umb, label='V/50', marker='o')
    plt.scatter(bmv_compl, umb_compl, label='V/36B', color='red', marker='o')
    plt.xlabel(r'(B-V) [mag]')
    plt.ylabel(r'(U-B) [mag]')
    plt.legend()
    plt.savefig(folder_fig + 'bmvVSumb.png')

# ----------------------------------------------------------------------------
```

```
    # Plot R-I vs U-B
    plt.clf()
    plt.scatter(rmi, umb, label='V/50', marker='o')
    plt.scatter(rmi_compl, umb_compl, label='V/36B', color='red', marker='o')
    plt.xlabel(r'(R-I) [mag]')
    plt.ylabel(r'(U-B) [mag]')
    plt.legend()
    plt.savefig(folder_fig + 'rmiVSumb.png')

# ------------------------------------------------------------------------------
    # Plot B-V vs R-I
    plt.clf()
    plt.scatter(bmv, rmi, label='V/50', marker='o')
    plt.scatter(bmv_compl, rmi_compl, label='V/36B', color='red', marker='o')
    plt.xlabel(r'(B-V) [mag]')
    plt.ylabel(r'(R-I) [mag]')
    plt.legend()
    plt.savefig(folder_fig + 'bmvVSrmi.png')

# ------------------------------------------------------------------------------
    # Plot B-V vs vsini
    plt.clf()
    plt.scatter(bmv, vsini, label='V/50', marker='o')
    plt.scatter(bmv_compl, vsini_compl, label='V/36B', color='red', marker='o')
    plt.xlabel(r'(B-V) [mag]')
    plt.ylabel(r'$v \sin i$ [km/s]')
    plt.legend()
    plt.savefig(folder_fig + 'bmvVSvsini.png')

# ==============================================================================

    create_txt_file(a=hd_selected, b=bmv, c=umb, d=rmi, e=vsini,
                    f=err_bmv, g=err_umb, h=err_vsini, i=companions.data,
                    j=obs_iue_date, l=sptyp_selected,
                    file_name=commum_folder + 'selected_stars.txt')

    create_txt_file_compl(a=hd_selected_compl, b=bmv_compl, c=umb_compl,
                          d=rmi_compl, e=vsini_compl, f=err_vsini_compl,
                          g=obs_iue_date_compl, h=sptyp_selected_compl,
                          file_name=commum_folder + 'selected_stars_compl.txt')

# ==============================================================================
    # example
    if False:
        R = np.array((data['Vc'] * 1e5) ** 2 /
                     10 ** data['logg'] / phc.Rsun.cgs)
        L = phc.sigma.cgs * np.array(data['Teff'], dtype=float)**4 * 4 *\
            np.pi * (R * phc.Rsun.cgs)**2 * phc.Lsun.cgs
        M = np.array((data['Vc'] * 1e5)**2 * (R * phc.Rsun.cgs) /
                     phc.G.cgs / phc.Msun.cgs)

# ==============================================================================
if __name__ == '__main__':
    main()
```

CHAPTER

SEVEN

REFERÊNCIAS E INDEX

7.1 Referências

Secchi, A.1866, Astronomische Nachrichten, 68, 63

Rivinius, T., Carciofi, A. C., & Martayan, C.2013, *Astronomy and Astrophysics Review*, 21, 69

b

BeFaVOr_web, 9

B

BeFaVOr_web (module), 9

C

create_list_files() (in module BeFaVOr_web), 9
create_txt_file() (in module BeFaVOr_web), 9

F

find_regular_expression() (in module BeFaVOr_web), 9

G

get_attribute() (in module BeFaVOr_web), 9
getTitle() (in module BeFaVOr_web), 9

I

iue_submission() (in module BeFaVOr_web), 9

P

plot_gal() (in module BeFaVOr_web), 10

R

read_simbad_coodr() (in module BeFaVOr_web), 10
read_simbad_data() (in module BeFaVOr_web), 10
read_txt() (in module BeFaVOr_web), 10
retrieve_ebmv_value() (in module BeFaVOr_web), 10

S

selecting_data() (in module BeFaVOr_web), 10
show_page_code() (in module BeFaVOr_web), 10

T

there_is_a_title() (in module BeFaVOr_web), 10

U

untar() (in module BeFaVOr_web), 11
unzip() (in module BeFaVOr_web), 11