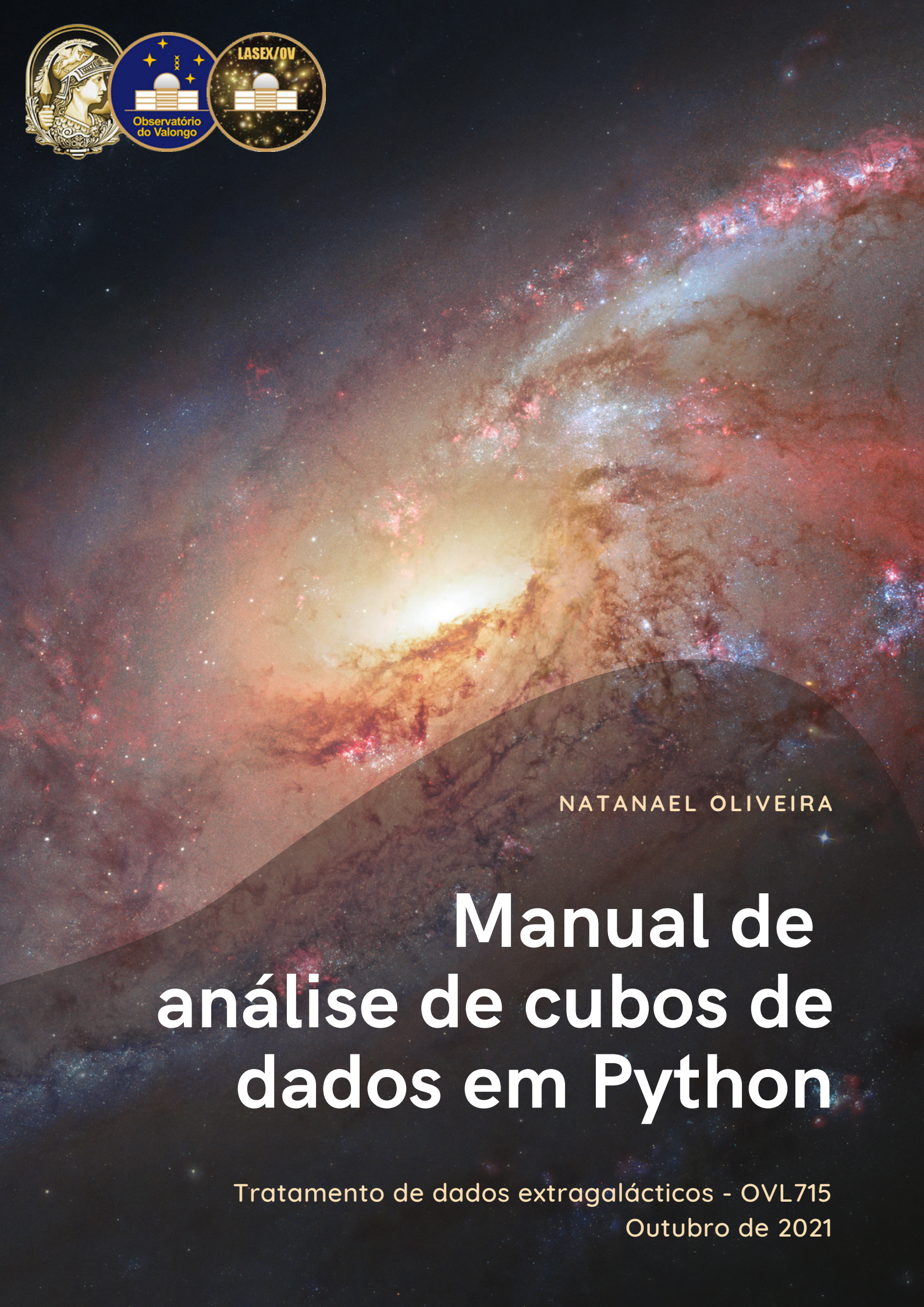


NATANAEL OLIVEIRA

Manual de análise de cubos de dados em Python

Tratamento de dados extragalácticos - OVL715

Outubro de 2021

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS MATEMÁTICAS E DA NATUREZA
OBSERVATÓRIO DO VALONGO
TRATAMENTO DE DADOS EXTRAGALÁCTICOS – OVL715

FEITO POR: Natanael Gomes de Oliveira
www.natanaeloliveira.com

Esse projeto foi realizado como parte da disciplina de Tratamento de Dados Extragalácticos, da pós-graduação em Astronomia do Observatório do Valongo.
Professores: Karín Menéndez-Delmestre & Thiago Signorini Gonçalves

Novembro 2021

Conteúdo

1 O que são cubos de dados? 4

2 Acessando os cubos de dados 6

2.1 VIVA Survey 6

2.2 Baixando os cubos de dados 8

3 Gerando os mapas de momento 9

3.1 Importando bibliotecas e pacotes necessários 9

3.2 Abrindo e visualizando o cubo de dados 10

3.3 Gerando os mapas de momentos 10

3.4 Plotando os mapas de momentos 11

3.5 Aplicando técnicas de otimização dos mapas de momentos 13

3.5.1 Suavização Gaussiana (Gaussian Smoothing) 14

3.5.2 Corte na razão Sinal-Ruído 15

3.6 Salvando os mapas de momentos 16

1. O que são cubos de dados?

Em diferentes áreas da ciência, especialmente em astronomia, é natural termos de lidar com o que chamamos de cubo de dados, ou “data cubes”, em inglês.

Um cubo de dados refere-se a um intervalo de valores em três ou mais dimensões que geralmente são usados para explicar a sequência de tempo dos dados de uma imagem. De forma geral, podemos dizer que esses cubos são usados para representar dados que são muito complexos para serem descritos por uma tabela de linhas e colunas.

Hoje em dia, todos os principais telescópios novos geram cubos de dados como seus produtos de dados primários, a exemplo do ALMA, ELVA, ASKAP, MeerKAT, LOFAR e SKA.

Abaixo, vemos um exemplo esquemático de um cubo de dados, o qual é proveniente do levantamento HI Parkes All-Sky Survey (HIPASS).

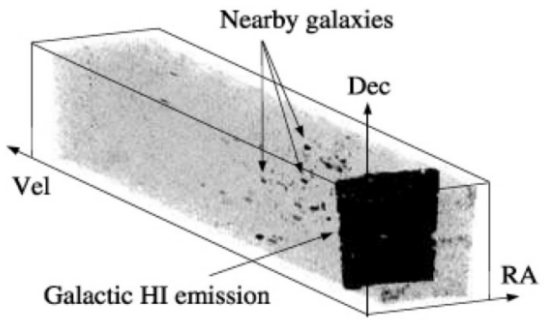
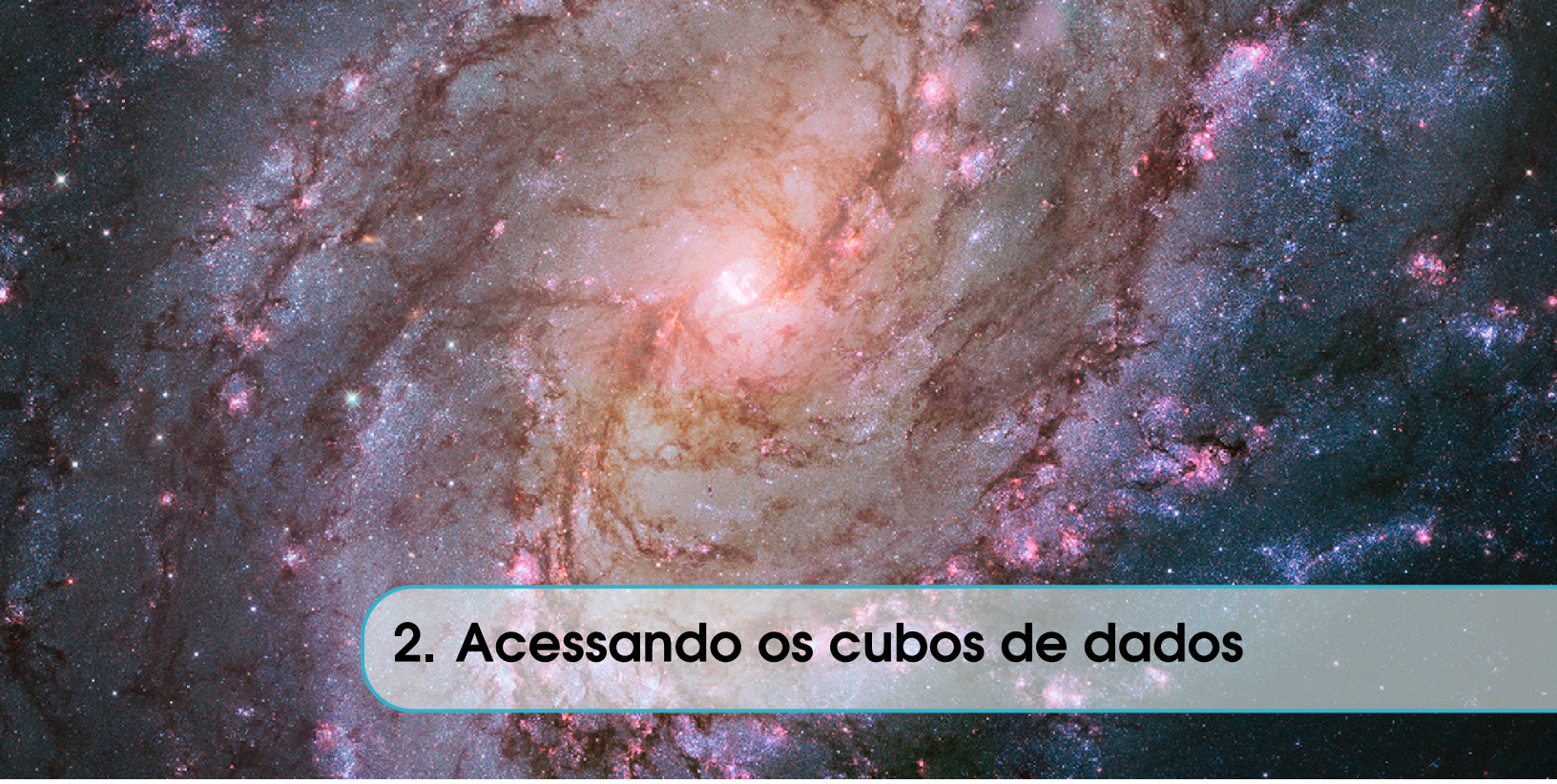


Figura 1.1: Esquema de cubo de dados mostrando galáxias próximas (pontos escuros) e o plano galáctico (folha escura).

O levantamento HIPASS é um levantamento da distribuição de hidrogênio atômico (HI), na banda do rádio do espectro eletromagnético (Meyer et al. 2004)[1]. No caso da Figura 1.1, vemos que, além das coordenadas espaciais, i.e., ascensão reta e declinação, temos ainda informações da emissão de HI e da velocidade rotacional deste gás. O acesso a essas informações torna os cubos de dados ferramentas essenciais para construção de mapas de intensidade, velocidade e dispersão de velocidade de galáxias, o que, por sua vez, nos permite estudar a distribuição da massa dinâmica – essencial para traçar a matéria escura.

[1] Disponível em: `https://ui.adsabs.harvard.edu/abs/2004MNRAS.350.1195M/abstract`



2. Acessando os cubos de dados

Para esse tutorial, usaremos os cubos de dados do levantamento VLA Imaging of Virgo in Atomic gas (VIVA) (Chung et al. 2009)[1]. Dessa forma, nada mais justo do que introduzirmos um pouco deste levantamento, mostrando algumas de suas características e informações importantes.

2.1 VIVA Survey

O VIVA Survey é um levantamento de imagens em HI com resolução de $\sim$ 15" em 53 galáxias do aglomerado de Virgem. Destas, 48 são galáxias espirais ou de morfologia semelhante à espiral e 5 objetos considerados irregulares.

Demais informações sobre o levantamento VIVA podem ser diretamente encontradas em seu site: ⟨`http://www.astro.yale.edu/viva/`⟩. Informações mais técnicas e de como acessar os dados de VIVA serão melhor discutidas ao longo deste manual.

[1] Disponível em: `https://ui.adsabs.harvard.edu/abs/2009AJ....138.1741C/abstract`

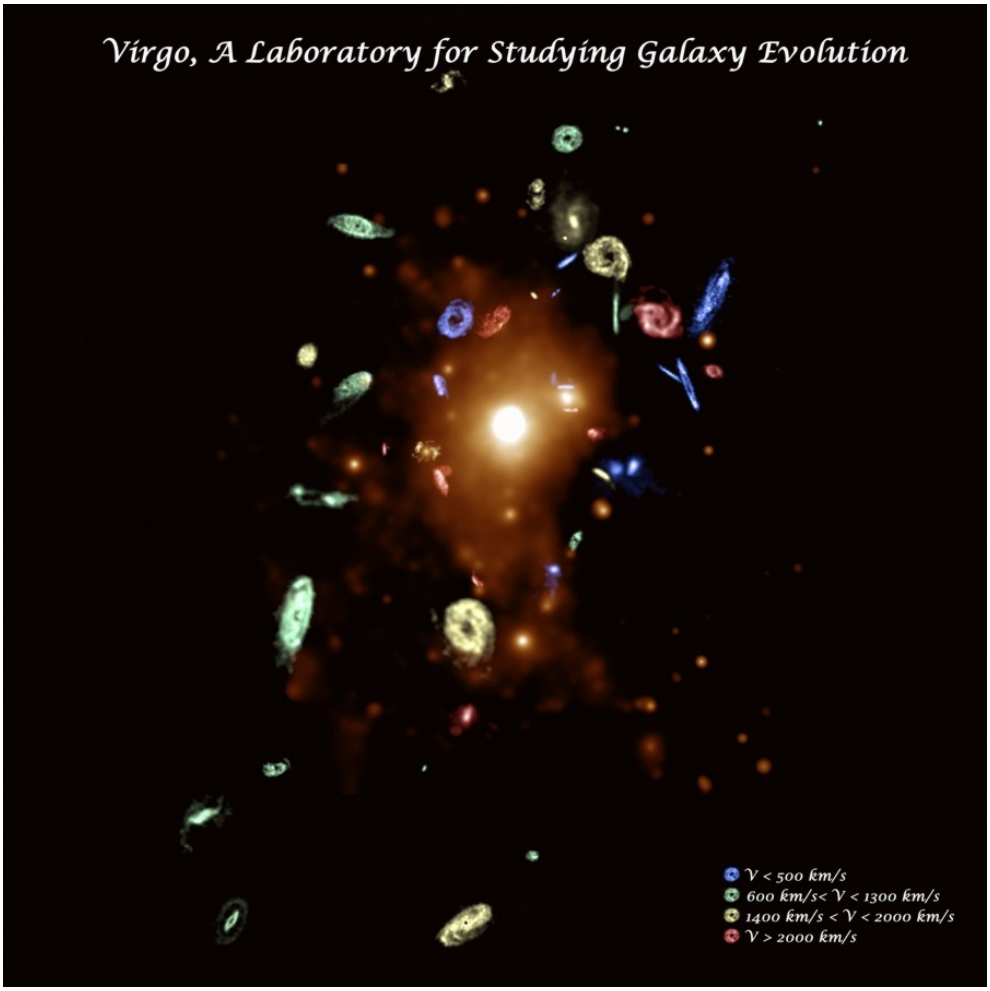
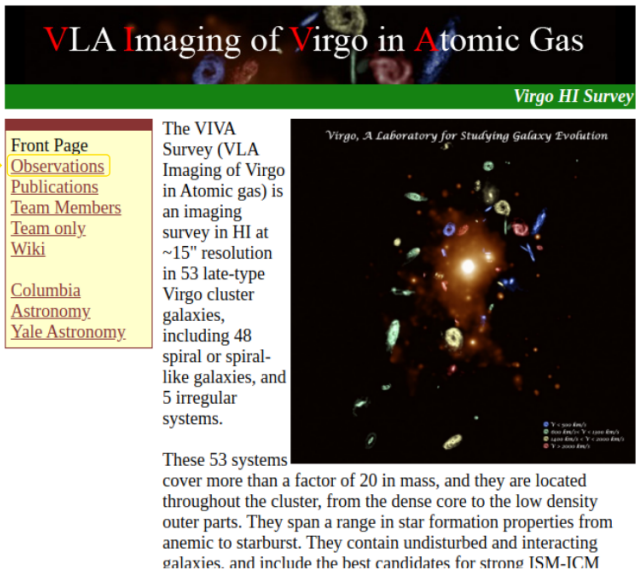


Figura 2.1: Imagem reunindo todos os objetos presentes no levantamento VIVA. As galáxias do levantamento estão apresentadas com diferentes cores, as quais dizem respeito à velocidade sistêmica de cada objeto. Disponível em: ⟨`http://www.astro.yale.edu/viva/32-Virgo_med.jpg`⟩.

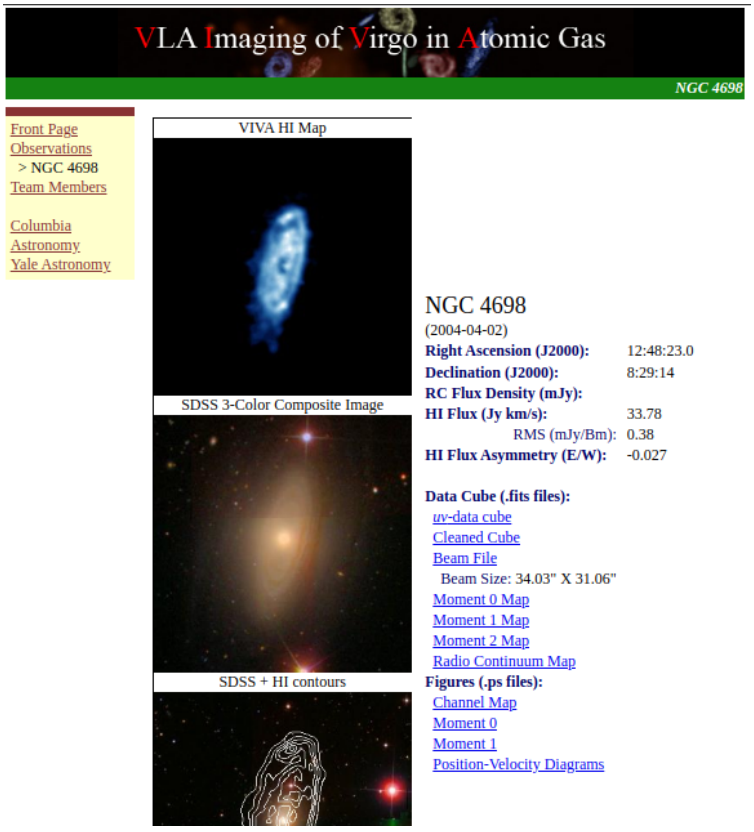
2.2 Baixando os cubos de dados

Os cubos de dados do levantamento VIVA (ou do levantamento que você esteja interessado em trabalhar) geralmente estão disponíveis no próprio site do levantamento.

No nosso caso, é necessário apenas abrir o site do levantamento (`http://www.astro.yale.edu/viva/`) e abrir a página "Observations", como mostrado ao lado:



É nesta janela que está a lista de todas as galáxias presentes no levantamento, além de informações como a data em que os dados foram produzidos. Ao clicar em um objeto, você será redirecionado à página que disponibiliza os cubos e vários outros dados relativos ao objeto selecionado. Aqui, focaremos nossa análise na galáxia NGC 4698.



Como podemos ver, esta janela nos fornece inúmeras informações, entre as quais irei chamar atenção às seguintes: coordenadas de ascensão reta e declinação do objeto; o "Cleaned Cube" que usaremos para gerar os mapas de momentos; os mapas de momentos 0, 1 e 2; diagramas de posição-velocidade; mapa de HI do VIVA e a imagem da galáxia vista como uma composição do levantamento SDSS + contornos em HI do VIVA Survey. Os mapas de momento são mapas que mostram o fluxo (momento 0), velocidade orbital do gás hidrogênio atômico (momento 1) e a velocidade de dispersão deste gás (momento 2). Para fazer o download do cubo que analisaremos, basta clicar em cima de "Cleaned Cube".

3. Gerando os mapas de momento

Neste capítulo, iremos mostrar o passo a passo para a construção dos mapas de intensidade e velocidade. A linguagem de programação utilizada neste manual é PYTHON[1].

3.1 Importando bibliotecas e pacotes necessários

Em um compilador de PYTHON de sua preferência, importe as seguintes bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

import astropy.units as u
from astropy.utils.data import download_file
from astropy.io import fits

from astropy.utils import data
data.conf.remote_timeout = 60

from spectral_cube import SpectralCube
from astropy import stats

from astroquery.esasky import ESASky
from astroquery.utils import TableList
from astropy.wcs import WCS
from reproject import reproject_interp
```

- A primeira biblioteca importada, o NUMPY, é usada principalmente para realizar cálculos em arrays multidimensionais. O NUMPY fornece um grande conjunto de funções e operações de biblioteca que ajudam a executar facilmente cálculos numéricos;
- A biblioteca MATPLOTLIB é uma biblioteca usada para a visualização de dados em Python, sendo muito utilizada no momento de plotar gráficos e figuras;
- O ASTROPY.UNITS é muito eficiente quando lidamos com cubos de dados, uma vez que este pode ser usado na definição, conversão e realização de operações aritméticas com

[1] Mais informações em: `https://www.python.org/`

quantidades físicas, como metros, segundos, hertz, jansky, etc., além de lidar também com unidades logarítmicas, como magnitude – bastante utilizada em astronomia.

Ao longo do programa, as especificidades de cada pacote utilizado serão melhor discutidas.

3.2 Abrindo e visualizando o cubo de dados

Felizmente, a biblioteca ASTROPY tem o pacote *spectral_cube*. Este pacote é realmente muito eficiente quando estamos trabalhando com cubo de dados, fazendo muitos trabalhos essenciais para manipularmos esses dados e até mesmo examiná-los rapidamente. Então, vamos ler o nosso cubo de dados como um SpectralCube!

```
hi_data = fits.open("ngc4698.cube.fits")
cube = SpectralCube.read(hi_data)
hi_data.close()
```

```
WARNING: StokesWarning: Cube is a Stokes cube, returning spectral cube for I component [spectral_cube.io.core]
```

Aqui, chamo atenção ao fato de que a variável *cube* tem os dados usando *spectral_cube*, enquanto que *hi_data* é o cubo de dados do arquivo .fits sem a formatação especial do SpectralCube.

Agora, vamos verificar como esse cube se apresenta:

```
print(cube)
```

```
SpectralCube with shape=(63, 512, 512) and unit=Jy / beam:
 n_x:    512  type_x: RA---SIN  unit_x: deg    range:   191.736000 deg:  192.454261 deg
 n_y:    512  type_y: DEC--SIN  unit_y: deg    range:     8.131500 deg:    8.841228 deg
 n_s:     63  type_s: FREQ      unit_s: Hz     range: 1414162179.795 Hz:1417189523.545 Hz
```

Podemos ver que nosso cubo de dados possui um eixo de Longitude Galáctica (n_x), Latitude Galáctica (n_y) e um eixo espectral em termos de frequência, listado como (n_s). Quando fazemos *print(cube)*, podemos ver a forma, tamanho e unidades de todos os eixos, bem como o intervalo de dados armazenados no cubo. Podemos ver, portanto, que os eixos espaciais estão em graus e o eixo espectral está em Hertz.

3.3 Gerando os mapas de momentos

Os mapas de momentos são ferramentas de análise muito úteis para estudar cubos de dados. De maneira geral, um "momento" é uma integral ponderada ao longo de um eixo (normalmente o eixo espectral), o qual pode fornecer informações sobre a intensidade total, velocidade média ou dispersão da velocidade ao longo das linhas de visão.

Para fazermos esses mapas, o *spectral_cube* torna isso muito simples com o método "*.moment()*". Como nosso eixo espectral tem unidade de frequência, podemos alterar esta unidade através do *.with_spectral_unit()*, de modo a ficarmos com o eixo espectral em termos de velocidade e, assim, conseguirmos gerar os mapas de momento.

```
moment_0 = cube.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=0)
```

```
moment_1 = cube.with_spectral_unit(u.Hz).moment(order=1)
```

Verificando as unidades:

```
print('Moment_0 tem a seguinte unidade: ', moment_0.unit)
print('Moment_1 tem a seguinte unidade: ', moment_1.unit)
```

```
Moment_0 tem a seguinte unidade:  Jy km / (beam s)
Moment_1 tem a seguinte unidade:  Hz
```

Dado um cubo espectral, é fácil extrair um subcubo cobrindo apenas um subconjunto do intervalo original no eixo espectral. Para fazer isso, você pode usar o método *spectral_slab()*. Este método leva os limites inferior e superior do eixo espectral, bem como uma frequência de repouso (opcional), e retorna uma nova instância SpectralCube. Os limites podem ser especificados como frequência, comprimento de onda ou velocidade, mas as unidades devem corresponder ao tipo das unidades espectrais no cubo.

```
slab = cube.spectral_slab(1414162179.795 * u.Hz, 1417189523.545 * u.Hz)
masked_slab = cube.with_mask(cube > -1.5332716e-07 * cube.unit)
```

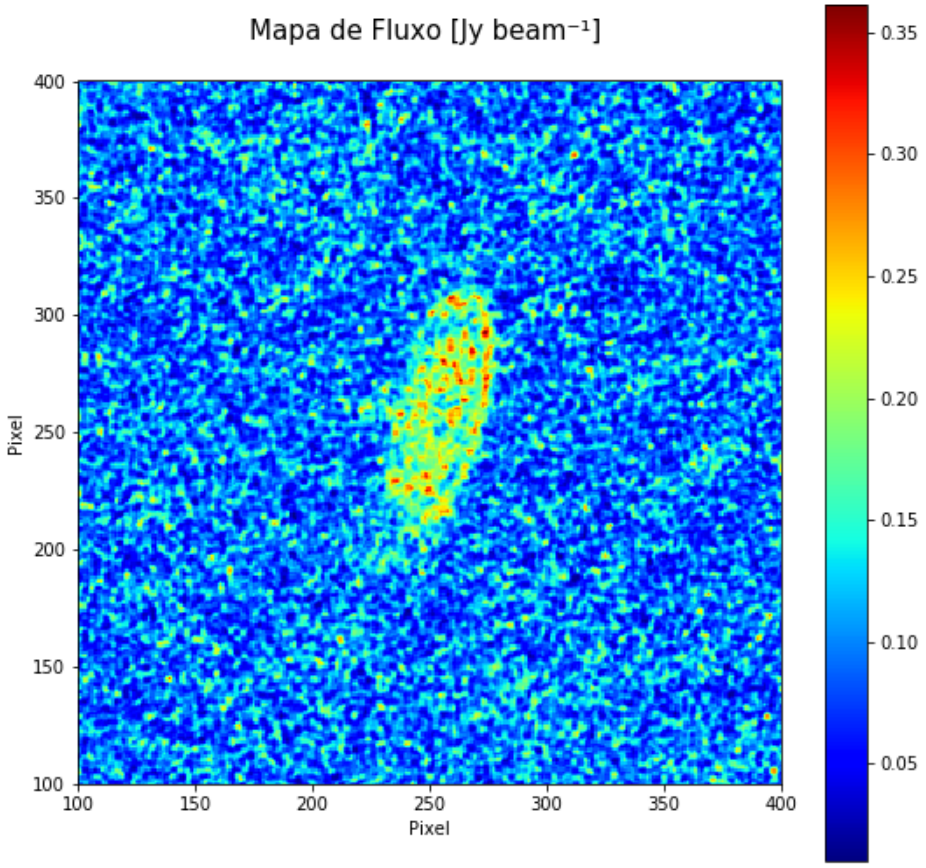
3.4 Plotando os mapas de momentos

Começaremos plotando o momento 0, que mostra a intensidade, como falado anteriormente:

```
fig = plt.figure()
fig.patch.set_facecolor('xkcd:white')
plt.figure(figsize=(9,9))
plt.imshow(masked_slab.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=0).hdu.data,
           cmap= 'jet')
plt.title("Mapa de Fluxo [Jy beam⁻¹] \n", fontsize=15)
plt.xlabel('Pixel')
plt.ylabel('Pixel')
plt.colorbar()
plt.xlim(100,400)
plt.ylim(100,400)
fig = plt.gcf()
plt.show()

# Obs.: Para salvar esse gráfico, descomente o seguinte comando:
# fig.savefig('NGC 4698 mapa de fluxo.png', format='png')
```

A última linha dessa célula de código mostra uma forma simples e prática de salvar os plots. Com o *fig.savefig()* é possível definir o título da imagem, assim como o seu formato, isto é, .png, .jpg, entre outros.

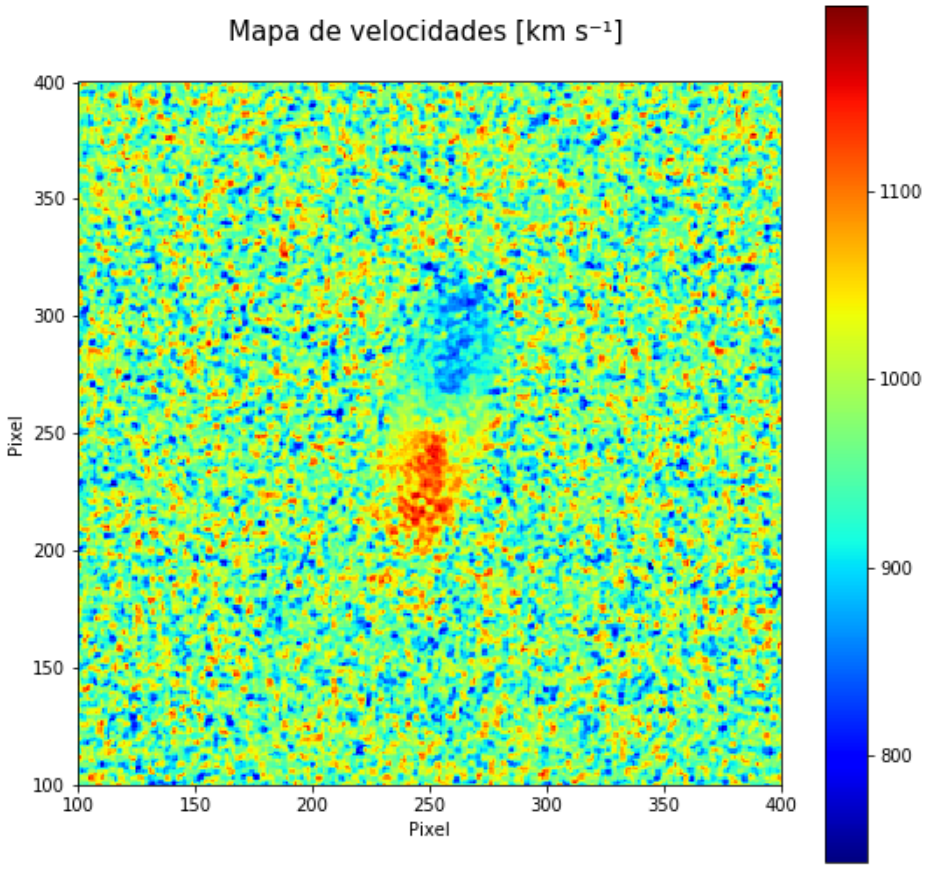
O plot da intensidade, então, fica assim:



Agora, podemos fazer um procedimento análogo para plotarmos o mapa de velocidades:

```
fig = plt.figure()
fig.patch.set_facecolor('xkcd:white')
plt.figure(figsize=(9,9))
plt.imshow(masked_slab.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=1).hdu.data,
           cmap= 'jet')
plt.title("Mapa de velocidades [km s⁻¹] \n", fontsize=15)
plt.xlabel('Pixel')
plt.ylabel('Pixel')
plt.colorbar()
plt.xlim(100,400)
plt.ylim(100,400)
fig = plt.gcf()
plt.show()

# Obs.: Para salvar esse gráfico, descomente o seguinte comando:
fig.savefig('NGC 4698 mapa de velocidades.png', format='png')
```

Ficando com o seguinte plot:



3.5 Aplicando técnicas de otimização dos mapas de momentos

Às vezes, os dados têm picos que são claramente artefatos do processamento ou devidos a alguma outra fonte externa. Em muitos casos, há "picos" de informação nos dados de frequência analisados, os quais não são provenientes da fonte observada, tornando os dados enviesados. Aqui, aplicaremos técnicas para podermos diminuir os efeitos de dados intrusos e aliviar esse problema. Iremos focar nossa análise no momento 1, isto é, no mapa de velocidades.

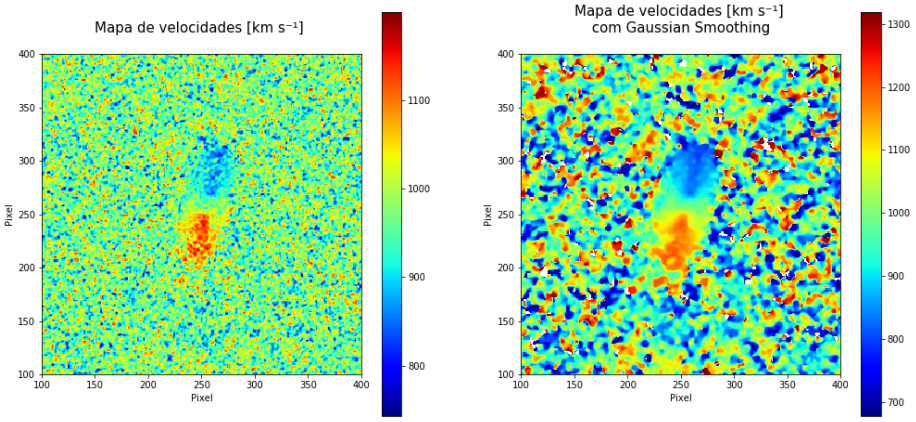
3.5.1 Suavização Gaussiana (Gaussian Smoothing)

Para fazermos essa suavização gaussiana, utilizar o AstroPy torna o processo muito mais simples, uma vez que as rotinas *spatial_smooth()* e *spectral_smooth()* nos permitem aplicar a técnica de suavização tanto na direção espacial quanto na direção espectral, respectivamente. Isso pode ser feito a partir dos comandos mostrados abaixo:

```
kernel = Gaussian2DKernel(x_stddev=2.5)
new_cube = cube.spatial_smooth(kernel)

kernel = Gaussian1DKernel(2.5)
new_cube = new_cube.spectral_smooth(kernel)
```

Podemos ver que a variável *"new_cube"* permanece sendo um cubo, mas agora com as direções espacial e espectral suavizadas. É importante ter em mente que rodar esses comandos de suavização gaussiana pode levar alguns minutos.

Após a suavização gaussiana, ficamos com o seguinte resultado:



O gráfico acima pode ser plotado através da seguinte célula de código:

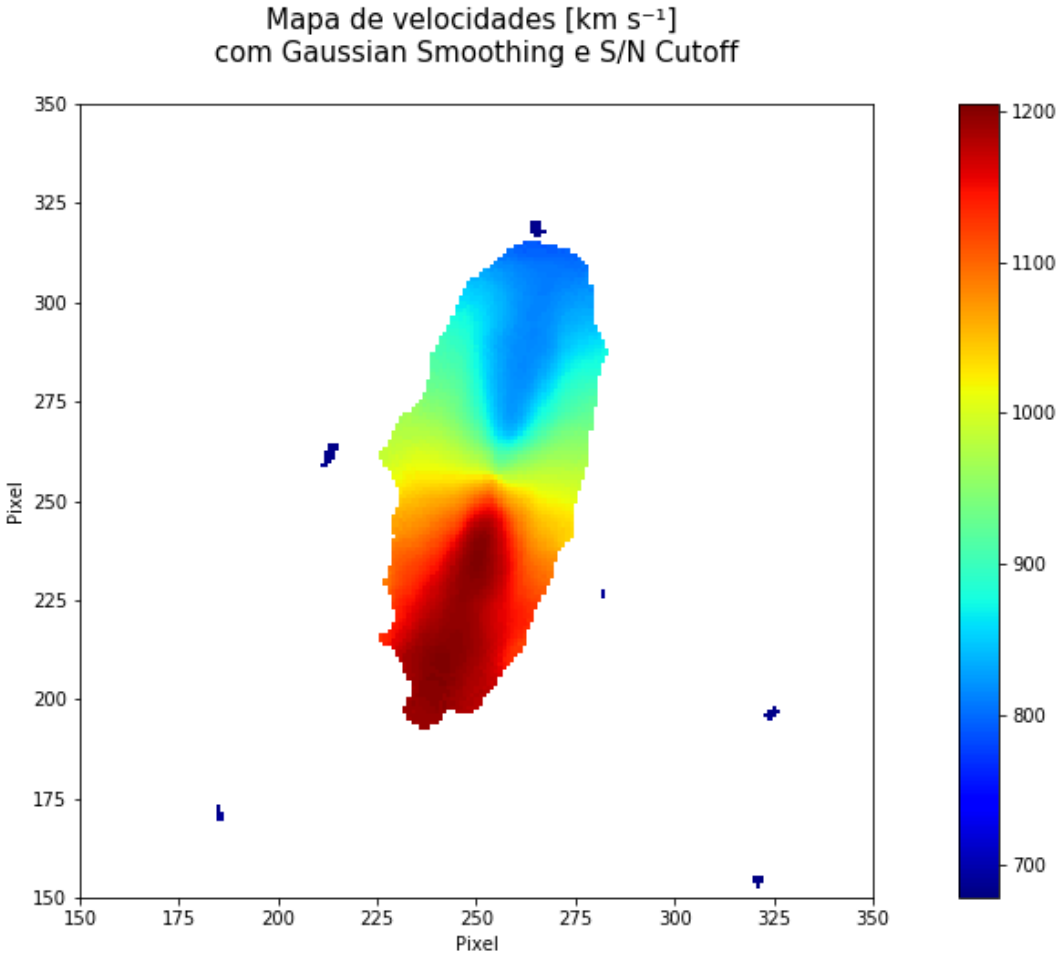
```
fig = plt.figure()
plt.figure(figsize=(17,8))

plt.subplot(1, 2, 1)
fig.patch.set_facecolor('xkcd:white')
# plt.figure(figsize=(9,9))
plt.imshow(masked_slab.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=1).hdu.data,
           cmap= 'jet')
plt.title("Mapa de velocidades [km s⁻¹] \n", fontsize=15)
plt.xlabel('Pixel')
plt.ylabel('Pixel')
plt.colorbar()
plt.xlim(100,400)
plt.ylim(100,400)
fig = plt.gcf()


plt.subplot(1, 2, 2)
fig.patch.set_facecolor('xkcd:white')
# plt.figure(figsize=(9,9))
plt.imshow(new_masked_slab.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=1).hdu.data,
           cmap= 'jet')
plt.title("Mapa de velocidades [km s⁻¹] \n com Gaussian Smoothing e S/N Cutoff \n", fontsize=15)
plt.xlabel('Pixel')
plt.ylabel('Pixel')
plt.colorbar()
plt.xlim(100,400)
plt.ylim(100,400)
fig = plt.gcf()

plt.show()

# Obs.: Para salvar esse gráfico, descomente o seguinte comando:
fig.savefig('plot.png', format='png')
```

3.5.2 Corte na razão Sinal-Ruído

Apesar do resultado obtido após a suavização gaussiana já mudar bastante a cara do mapa de velocidades, um corte no sinal-ruído ainda se mostra necessário, uma vez que a imagem permanece significativamente afetada por sinais não provenientes do objeto que estamos interessados. Sendo assim, uma estratégia muito eficaz nesse caso é fazer um corte na razão sinal-ruído. Para isso, devemos pegar uma região de céu da imagem (e ter cuidado para que esta região não esteja sendo afetada por emissões provenientes da galáxia estudada) e verificar sua estatística. Como corte, usaremos apenas os dados que possuem razão sinal-ruído maior que 7, isto é, o sinal é pelo menos sete vezes maior que o ruído.

```
new_slab = new_cube.spectral_slab(1414162179.795 * u.Hz, 1417189523.545 * u.Hz)
new_masked_slab = new_cube.with_mask(new_cube > 7*ruido * new_cube.unit)
```

Fazendo isso, ficamos com o mapa de velocidades final (momento 1):



3.6 Salvando os mapas de momentos

Essas novas projeções bidimensionais podem ser salvas como novos arquivos .fits da seguinte forma:

```
moment1 = new_masked_slab.with_spectral_unit(u.km/u.s, velocity_convention='radio').moment(order=1)
moment1.write('moment1_NGC4698.fits')
```

Onde a variável *"moment1"* armazena os dados do mapa final de velocidades, após a aplicação da suavização gaussiana e corte por sinal-ruído.