TP8_CPES2

September 26, 2022

1 CPES 2 : TP8 - Traitement de données spectroscopiques

Année 2022-2023

Noms :

Prénoms :

1.1 1 - Objectif

Le but de cette séance est d'étudier des spectres observés d'étoiles et de planètes.

Ce notebook fait office de compte-rendu. Vous compléterez le code et rédigerez vos réponses directement dessus, afin de rendre un notebook par groupe à la fin de la séance.

Pensez à renommer le nom de ce notebook avec vos noms et prénoms sous la forme : NOM_PRENOM_TP8_CPES2.ipynb

Ne pas hésiter à ouvrir votre notebook de la séance précédente afin de vous aider !

1.2 2 - Qualité des données obtenues

1.2.1 2.1 - Chargement des librairies python

La première étape consiste à charger les différents modules python dont nous aurons besoin tout au long de ce TP : - *os* (https://docs.python.org/3/library/os.html) dispose d'outils pour naviguer dans le système et lister les fichiers d'un dossier de l'ordinateur - *numpy* (https://numpy.org/doc/stable/) permet de manipuler des tableaux de données à plusieurs dimensions et d'effectuer toute sorte d'opérations (sommes, multiplications matricielle, moyennes…) - *scipy* (https://scipy.org/) ajout de nombreux outils mathématiques pour les applications scientifiques - *matplotib* (https://matplotlib.org/stable/index.html) offre un large panel d'outils permettant de visualiser les données et de produire des graphiques - *astropy* (https://docs.astropy.org/en/stable/index.html) est une librairie dédiée à l'astronomie et l'astrophysique et contient une multitude d'outils pratiques pour le traitement des données. Nous utilisons surtout ici la fonction de lecture des fichiers au format FITS.

Note : ne pas hésiter à consulter la documentation de ces différents modules en cas de besoin. De nombreux exemples d'utilisation y sont présentés.

[ ]:
```
# Importation des modules
import os
import numpy as np                # on peut renommer un module avec une variable␣
 ↪plus courte avec le mot clé "as"
import matplotlib.pyplot as plt
import ipywidgets as wdg          # ipython notebook widgets

from astropy.io import fits       # "from" permet de ne charger que le␣
 ↪sous-module "fits" de la librairie "astropy" afin de ne pas encombrer␣
 ↪inutilement la mémoire
from scipy.interpolate import interp1d
from matplotlib.widgets import Cursor
from scipy.optimize import curve_fit

# commande "%magic" pour utiliser le backend "widget" de matplotlib permettant␣
 ↪d'avoir les outils interactifs (tels que le zoom) sur les figures
%matplotlib widget
```

Dans Jupyter Lab, une commande très utile est le **?** : placé derrière une commande, il permet d'accéder rapidement à sa documentation pour savoir comment l'utiliser. Exemple avec la fonction *print( )* :

[ ]:
```
print?
```

Il suffit donc de créer une nouvelle cellule, et d'écrire le nom de la fonction en remplaçant les parenthèse **( )** par le point d'interrogation **?** pour obtenir des informations sur l'utilisation de la fonction.

1.2.2 2.2 - Vérification des données

Nous commençons par charger nos données de la même façon que lors du TP précédent :

[ ]:
```
# récupération de la liste de fichiers dans le répertoir "./Data_TP_CPES2/
 ↪Donnees/" :
path = './Data_TP_CPES2/Donnees/'
file_list = os.listdir(path)

# chargement des fichiers fits dans un dictionnaire
donnees = {}
for file_name in file_list:
    try:
        current_fits = fits.open(path+file_name)[0] # fits.open() renvoie une␣
 ↪list de tableaux de données (appelés "HDU") contenus dans le fichier FITS.␣
 ↪Ici nos fichiers ne contiennent qu'un seul HDU, que l'on sélectionne avec␣
 ↪l'indice [0]
        donnees[file_name] = current_fits           # on stock ici le fichier .
 ↪fits chargé dans un dictionnaire afin d'y accéder plus facilement par la␣
 ↪suite en utilisant le nom du fichier d'origine
```

```
        print(f'{file_name:<80} has been loaded !')
    except Exception as e:                              # En cas d'echec de␣
↪chargement du fichier, la structure try:... except:... permet d'empêcher au␣
↪code de planter en affichant l'erreur et en passant au fichier suivant (par␣
↪exemple si le fichier n'est pas au format .fits)
        print(f'Can\'t load {file_name:<69} {e}')
```

Bref rappel de la séance précédente :

On rappelle que grâce à cette méthode de chargement des données, tous les spectres sont accessibles via leur nom dans le dictionnaire *données*. Par exemple pour charger les données des fichiers "dark_10s_1.fits" et "dark_10s_2.fits", on peut utiliser la commande suivante :

```
dark_10s_1 = donnees['dark_10s_3.fits'].data # le .data permet de récupérer les données
dark_10s_2 = donnees['dark_10s_3.fits'].data
```

On peut alors effectuer des opérations avec ces variables, calculer une moyenne par exemple :

```
dark_moy_10s = (dark_10s_1 + dark_10s_2) / 2
```

On peut consulter le header d'un fichier de la façon suivante :

```
print(donnees['dark_10s_1.fits'].header)
```

Enfin, on peut sauvegarder un spectre dans un fichier au format ".dat" de la façon suivante :

```
name = "nom_du_fichier.dat"
np.savetxt(name,dark_moy_10s)
print(f'{name} a bien été sauvegardé !')
```

Dans une démarche scientifique impliquant des mesures, l'une des premières choses à faire est de s'interroger sur la qualité des données, leur étalonnage, etc. Dans un premier temps, nous allons vérifier la qualité des étalonnages en longueur d'onde des données en prenant pour exemple le premier spectre de Sirius (sirius_2s_1).

Question 1 : À partir des trois darks disponibles associés au même temps de pose que sirius, réalisez un master dark. Détaillez les opérations que vous réalisez pour cela. Vous nommerez la variable "master_dark_2s" et enregistrez le résultat dans un fichier "nom_prenom_Q1_MASTER_DARK.dat".

[ ]:
```
# CODE A COMPLETER
```

Répondre ici :

Question 2 : Soustrayez le master dark au spectre de Sirius que vous avez chargé. **Stockez le résultat dans une variable "sirius_q2"** et enregistrez le résultat dans un fichier "nom_prenom_Q2_SIRIUS_.dat".

[ ]:
```
# CODE A COMPLETER
```

Question 3 : Le fichier de raies "PSL.ids" dans le dossier "Raies" contient les éléments les plus communs pour les étoiles. Le code ci-dessous permet de superposer un spectre d'étoile avec ce fichier de raies de référence.

Executez simplement les deux cellules ci-dessous, puis identifiez dans la figure les raies plus importantes et déterminez quel élément est prépondérant dans l'atmosphère de Sirius. Les données sont-elles bien étalonnées ?

[ ]:
```
# lecture du fichier 'PSL.ids' et remplissage du dictionnaire
raies_list = [] # liste contenant les raies du fichier 'PSL.ids' sous la forme␣
 ↪['element','position']
with open('./Data_TP_CPES2/Raies/PSL.ids','r') as f:
    for line in f.readlines()[10:]:
        # skip 10 first lines
        try:
            elements = line.replace('\t',' ').replace('*','').split(' ')
            element_name = elements[-1].replace('\n','')
            element_pos  = float(elements[0])/10 # conversion des Angströms en␣
 ↪nm
            raies_list.append([element_name,element_pos])
        except Exception as e:
            print(e)
```

[ ]:
```
# récupération des longueurs d'onde du spectre à partir du header
lmin = float(donnees['Sirius_2s_1.fits'].header['CRVAL1']) # longueur d'onde du␣
 ↪premier point en nm
nb   = int(donnees['Sirius_2s_1.fits'].header['NAXIS1'])   # nombre de canaux␣
 ↪d'échantillonnage
step = float(donnees['Sirius_2s_1.fits'].header['CD1_1'])  # écart entre deux␣
 ↪échantillon en nm

wave = np.linspace(lmin,lmin+nb*step,nb)

# Affichage du spectre de l'étoile et des position de raies
plt.figure()
plt.plot(wave,sirius_q2)

for element,position in raies_list:
    plt.vlines(position,sirius_q2.min(),sirius_q2.max(),color='k',lw=0.5)
    plt.text(position,0,element,rotation=90)
```

Répondre ici :

1.3 3 - Réduction de vos données

1.3.1 3.1 - Comparaison des spectres stellaires

Question 4 : Faites la réduction des données pour les étoiles Sirius, Tsih, Capella, Aldébaran et Bételgeuse, contenues dans la variable dictionnaire "donnees". Vous penserez bien à appliquer la

méthode de traitement des données que vous avez appliquée au TP précédent : 1. Création d'une moyenne des DARKS pour chaque temps de pose utilisé pour les spectres des étoiles (MASTER DARKS) et les FLATS 2. Création du MASTER FLAT, c'est-à-dire la valeur normalisée à 1 sur tout le spectre de la moyenne des FLATS auquel on a soustrait une moyenne des DARKS enregistrés avec le même temps de pose que les FLATS 3. Création d'une moyenne des spectres de l'étoile (MASTER ETOILE) 4. Création du spectre traité = (MASTER ETOILE - MASTER DARK)/(MASTER FLAT)

On aura ainsi autant de MASTER DARK que de temps de pose. Attention ainsi à soustraire pour chaque étoile le MASTER DARK correspondant à son temps de pose. N'oubliez pas d'enregistrer vos résultats (notamment MASTER FLAT et MASTER DARK).

[ ]:
```
# CODE A COMPLETER :
```

Complétez le code ci-dessous pour affichez les 5 spectres sur un même graphique. Faites en sorte qu'ils aient environ le même niveau en les normalisant par leur moyenne. Enregistrez ce graphique sous le nom nom_prenom_Q4_WIEN_ETOILES.png

Rappel : le code suivant permet d'ajouter un spectre à la figure :

```
plt.plot(x,y,label='nom') # avec x et y les coordonnées des points, et "nom" le nom correspond
```

[ ]:
```
plt.figure()
# CODE A COMPLETER

########################################

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel(f'Flux')
plt.legend()
plt.title('Comparaison des spectres d\'étoiles')
```

Faites valider l'allure de vos spectres par un encadrant avant de passer à la suite !!

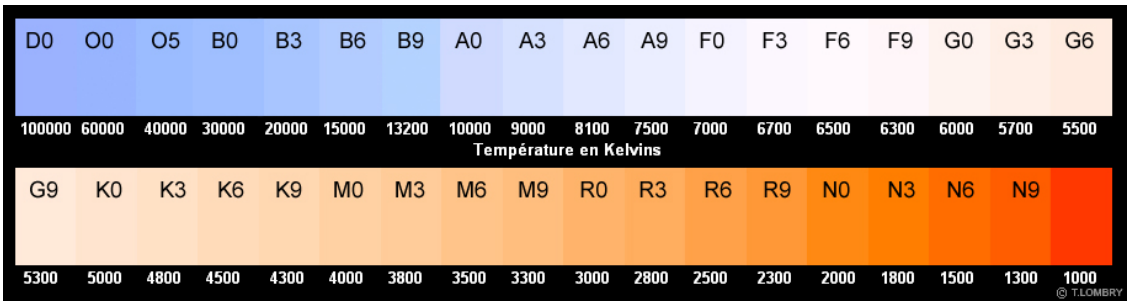
Question 5 : À partir des spectres de chaque étoile, identifiez max et calculez leurs températures T_{Wien} en utilisant la loi de Wien. Décrivez bien votre méthode de calcul, et utilisez le tableau 1 ci-dessous pour résumer vos résultats (double-cliquez dessus afin de modifier les valeurs).

Répondre ici :

Etoile	Tsih	Sirius	Capella	Aldébaran	Bételgeuse
λ_{max} (nm)					
T_{Wien} (K)					

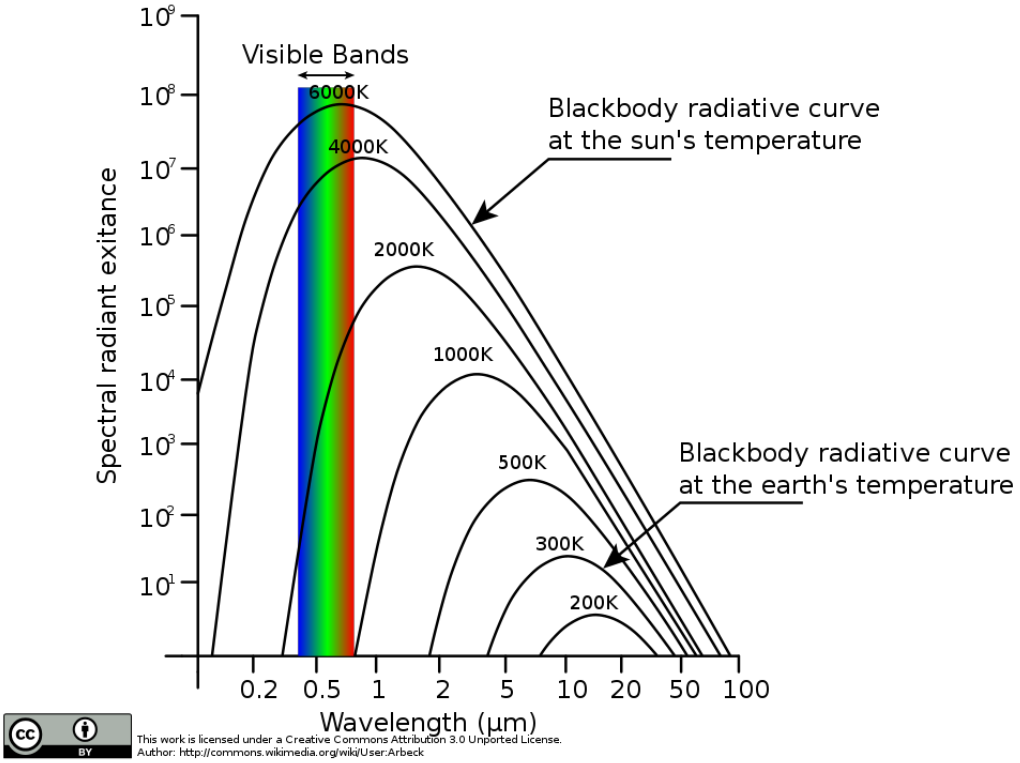
Question 6 : À l'aide de la table des types spectraux et des courbes de corps noir données ci-dessous, déterminez le type spectral de chaque étoile à partir de la température estimée pré cédemment. Entrez vos résultats dans la ligne 'type de spectre' du tableau ci-dessous.

Les types spectraux des étoiles en fonction de leur température.



Phrase mnémotechnique pour se souvenir des types spectraux : " **O**h **B**e **A** **F**ine **G**irl/**G**uy,**K**iss **M**e" !

Courbes de corps noir en fonction de la longueur d'onde pour plusieurs températures.



Répondre ici :

Etoile	Tsih	Sirius	Capella	Aldébaran	Bételgeuse
λ_{max} (nm)					
T_{Wien} (K)					
Type de spectre					

Question 7 : Le code ci-dessous permet de comparer vos spectres à ceux des spectres typiques d'étoiles (dans le dossier "Etoiles"). Remplacez simplement "aldebaran_normalise" par l'un de vos

spectres (et mettez à jour le “Label” pour que la légende corresponde) et relancez la cellule afin de comparer ce spectre avec les étoiles de référence.

Faites la comparaison uniquement entre 400 et 800 nm, là ou le spectromètre PSL est utilisable. Déduisez-en une autre estimation de la température et du type spectral de l'étoile. Résumez vos résultats dans les lignes ‘type de spectre (comparison)’ et ‘T est’ du tableau ci-dessous.

[ ]:
```
plt.figure()

# CODE A COMPLETER : affichage de l'un de vos spectres

##########################################################################################

# étoiles de références (code déjà complet)
ref_star_list = os.listdir('./Data_TP_CPES2/Etoiles/') # liste des fits
for file in ref_star_list:
    # chargement du fits
    star = fits.open('./Data_TP_CPES2/Etoiles/'+file)[1]
    spectra = star.data['FLUX']
    spectra /= np.mean(spectra) # normalisation
    star_wave = star.data['WAVELENGTH'] / 10 # convert to nm
    # affichage
    plt.plot(star_wave,spectra,label=file,lw=0.5)

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel(f'Flux')
plt.legend()
plt.title('Comparaison avec les étoiles de référence')
```

Répondre ici :

Etoile	Tsih	Sirius	Capella	Aldébaran	Bételgeuse
λ_{max} (nm)					
T_{Wien} (K)					
Type de spectre					
Type de spectre (comparaison)					
T_{est} (K)					

1.4 4 - Comparaison des spectres de Vénus, Mars, Uranus et Jupiter

Vous allez utiliser des observations de Vénus, Mars, Jupiter et Uranus contenues dans le dossier “Donnees” pour comparer les albédos de ces planètes. L'albédo est le pouvoir réfléchissant d'une surface, soit le rapport de l'énergie lumineuse réfléchie à l'énergie lumineuse incidente. C'est une grandeur sans dimension.

Pour aller plus vite, vous n'utiliserez qu'une seule image pour chaque planète si la qualité des données est suffisante. Vous n'oublierez pas d'y soustraire le DARK correspondant et de diviser par le MASTER FLAT que vous avez déjà calculé dans la partie précédente.

Le spectre de Jupiter est dans le répertoire Jupiter_soleil dans lequel vous trouverez également un spectre du Soleil. **Ces deux spectres sont déjà traités** (soustraction d'un MASTER DARK et division par un MASTER FLAT).

Question 8 : D'ou provient la lumière que l'on reçoit des planètes ?

Répondre ici :

Question 9 : Que proposez-vous pour estimer l'albédo de ces planètes ? L'albédo calculé ne peut l'être qu'à un coeffcient multiplicatif près dans notre cas. Pourquoi ?

Répondre ici :

Question 10 : On rappelle que tout les fichiers fits ont été chargés dans le tableau *donnees*, et sont donc accessibles via la commande *donnees["nom_du_fichier.fits"]* (en ajoutant un *.data* en fin de ligne pour accèder aux données).

Le code ci-dessous est déjà complet et permet de charger le spectre du Soleil dans une variable *soleil_final* et de calculer l'albedo de jupiter dans une variable *jupiter_albedo*. Executez simplement la cellule ci-dessous :

[ ]:
```
# chargement des spectres de Jupiter et du Soleil (code déjà complet)
jupiter_fits = fits.open('./Data_TP_CPES2/Jupiter_soleil/Spectre_jupiter.
 ↪fits')[0]
soleil_fits  = fits.open('./Data_TP_CPES2/Jupiter_soleil/Spectre_solaire.
 ↪fits')[0]

jupiter_final = jupiter_fits.data
soleil_final  = soleil_fits.data

jupiter_albedo = jupiter_final / soleil_final
jupiter_albedo = jupiter_albedo / np.mean(jupiter_albedo) # normalisation

# chargement des longueurs d'ondes correspondantes :
ref_px = jupiter_fits.header["CRPIX1"] # Ref pixel nb
lref   = jupiter_fits.header["CRVAL1"] # wavelength value at ref pixel
step   = jupiter_fits.header["CD1_1"]  # step in nm btw each pixel
nb     = jupiter_fits.header["NAXIS1"] # tot nb of pixels
lmin   = lref-ref_px*step # wavelength of first pixel
wave_jup_sol = np.linspace(lmin,lmin+nb*step,nb)

# reinterpolation des spectres de Jupiter et du Soleil sur la même échelle en␣
 ↪longueur d'onde que les autres données
finterp = interp1d(wave_jup_sol,jupiter_albedo,bounds_error=False,fill_value=1)
jupiter_albedo = finterp(wave)

finterp = interp1d(wave_jup_sol,soleil_final,bounds_error=False,fill_value=1)
soleil_final = finterp(soleil_final)
```

Complétez les cellules suivantes afin de calculer l'albédo des 4 planètes en divisant les spectres des planètes par le spectre du Soleil fourni (variable *soleil_final*) et en normalisant la moyenne de chaque albédo à 1 entre 450 et 700 nm. Observez-vous plutôt des raies atomiques fines ou des bandes d'absorption moléculaires assez élargies ? Pour quelles planètes ?

[ ]:
```
# CODE A COMPLETER : calcul de l'albedo de Venus, Mars et Uranus
```

[ ]:
```
# CODE A COMPLETER : affichage de l'albedo des 4 planètes
```

Répondre ici :

Faites valider l'allure de vos spectres par un encadrant avant de passer à la suite !!

Question 11 : Les spectres de deux de ces planètes sont présentés sur la figure ci-dessous en comparaison de spectres de nuages et d'océan pour la Terre. À quelles planètes correspondent les planètes n°1 et n°2 de cette figure ?

Albédos de planètes.

Répondre ici :

Question 12 : Dans les atmosphères des 2 planètes géantes (Uranus et Jupiter), il y a notamment du dihydrogène, de l'hélium, de l'ammoniac (NH3) et du méthane (CH4). Le code ci-dessous permet de superposer vos spectres avec les listes de raies du dossier "Raies". Executez la cellule, et déterminez quelles espèces parmi les 4 citées ci-dessus sont observées dans les spectres des 2 planètes géantes.

[ ]:
```
# lecture des listes de raies et remplissage du dictionnaire

raies_list = [] # liste contenant les raies du fichier 'PSL.ids' sous la forme␣
 ↪['element','position']

with open('./Data_TP_CPES2/Raies/CH4.ids','r') as f:
    for line in f.readlines()[10:]:
        # skip 10 first lines
        try:
            elements = line.replace('\t',' ').replace('*','').split(' ')
            element_name = elements[-1].replace('\n','')
            element_pos  = float(elements[0])
            raies_list.append([element_name,element_pos])
        except Exception as e:
            print(e)

with open('./Data_TP_CPES2/Raies/H_H2_He.ids','r') as f:
    for line in f.readlines()[10:]:
        # skip 10 first lines
        try:
            elements = line.replace('\t',' ').replace('*','').split(' ')
            element_name = elements[-1].replace('\n','')
```

```
            element_pos  = float(elements[0])/10 # conversion des Angströms en␣
 ↪nm
            raies_list.append([element_name,element_pos])
        except Exception as e:
            print(e)

with open('./Data_TP_CPES2/Raies/NH3.ids','r') as f:
    for line in f.readlines()[10:]:
        # skip 10 first lines
        try:
            elements = line.replace('\t',' ').replace('*','').split(' ')
            element_name = elements[-1].replace('\n','')
            element_pos  = float(elements[0])
            raies_list.append([element_name,element_pos])
        except Exception as e:
            print(e)
```

[ ]:
```
# CODE A COMPLETER : affichage de vos spectres

#################################################################

# Code déjà complet : affichage des éléments de référence
for element,position in raies_list:
    plt.vlines(position,5,0,color='k',lw=0.5)
    plt.text(position,0,element,rotation=90)

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel('Albedo normalisé')
plt.title('Albedo de Jupiter et Uranus')
plt.legend()
```

Répondre ici :

1.5 5 - Mesures des profils de raies stellaires

La forme des raies stellaires permet d'étudier les conditions de température et de pression dans les atmosphères stellaires. La qualité des mesures et la résolution spectroscopique est ici beaucoup trop faible pour faire une étude avancée. Néanmoins, il est possible de faire un certain nombre de remarques intéressantes.

Nous allons nous focaliser sur une raie centrée autour de 650 nm visible sur les étoiles Tsih (25 000 K), Zeta Tau (22 000K), Sirius (9900 K), Alcyone (13 000K) et Eta Cassiopée (5700 K).

La cellule suivant va charger les spectres de chaque étoile contenus dans le dossier "Partie4-5" dans un nouveau dictionnaire "etoiles_part5" et les afficher. Ces spectres ont déjà été étalonnés et sont normalisés aux mêmes flux autour de la raie intéressante.

[ ]:
```
# Chargement des spectres du dossier "Partie4-5"

# list des fichiers fits à charger
part5_file_list = [name for name in os.listdir('./Data_TP_CPES2/Partie4-5/') if␣
 ↪'.fits' in name]

# remplissage du dictionnaire avec les fits chargés
etoiles_part5 = {}
for file in part5_file_list:
    star_fits = fits.open('./Data_TP_CPES2/Partie4-5/'+file)[0]
    etoiles_part5[file] = star_fits

# récupération de l'échelle en longueur d'onde
ref_px = star_fits.header["CRPIX1"] # Ref pixel nb
lref   = star_fits.header["CRVAL1"] # wavelength value at ref pixel
step   = star_fits.header["CD1_1"]  # step in nm btw each pixel
nb     = star_fits.header["NAXIS1"] # tot nb of pixels
lmin   = lref-ref_px*step # wavelength of first pixel
wave_part5 = np.linspace(lmin,lmin+nb*step,nb)

# Affichage des spectres
plt.figure()
for star_name,star_fits in etoiles_part5.items():
    plt.plot(wave_part5,star_fits.data,label=star_name)

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel('Flux')
plt.legend()
plt.title('Comparaison des raies')
```

On accèdera donc dans toute la suite au spectre d'une étoile via la commande suivante (exemple pour Tsih) :

```
Tsih_spectra     = etoiles_part5['Tsih.fits'    ].data
```

Question 13 : À quelle longueur d'onde se situe cette raie exactement ? À quel élément cela correspond-il ? À quelle longueur d'onde théorique devrait se situer cette raie ?

Répondre ici :

Nous allons utiliser uniquement la partie du spectre située entre 645 nm et 665 nm. Pour cela, nous allons créer un **masque** avec *numpy* permettant de sélectionner les spectres uniquement dans cette région. Un masque s'applique simplement à un vecteur de données en l'ajoutant sous la forme [masque] (entre crochets) après les données, comme dans l'exemple ci-dessous :

[ ]:
```
masque = np.logical_and(wave_part5 > 645, wave_part5 < 665) # création du␣
 ↪masque : permet de ne sélectionner que les données aux longueurs d'ondes␣
 ↪entre 645 et 665nm
```

```
# Affichage des spectres
plt.figure()
for star_name,star_fits in etoiles_part5.items():
    # en ajoutant [masque] après le vecteur des longueurs d'ondes et des␣
 ↪données, on ne sélectionne que les données vérifiant la condition du masque
    plt.plot(wave_part5[masque],star_fits.data[masque],label=star_name)

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel('Flux')
plt.legend()
plt.title('Comparaison des raies entre 645 et 665nm')
```

On voit dans le graphique ci-dessus que le continuum n'est pas constant, mais penché. Pour la suite, nous aurons besoin de redresser le continuum horizontalement à 1. Pour ce faire, la cellule ci-dessous vous permet de réaliser un ajustement du continuum, c'est-à-dire de trouver une fonction (une simple droite dans le cas présent) dont la forme se rapproche raisonnablement du continuum afin de le ramener à 1 par une division. **Executez simplement la cellule sans la modifier.**

[ ]:
```
# Affichage des spectres
fig = plt.figure()
for star_name,star_fits in etoiles_part5.items():
    # en ajoutant [masque] après le vecteur des longueurs d'ondes et des␣
 ↪données, on ne sélectionne que les données vérifiant la condition du masque
    plt.plot(wave_part5[masque],star_fits.data[masque],label=star_name)

plt.xlabel(f'$\lambda$ [nm]')
plt.ylabel('Flux')
plt.legend()
plt.title('Comparaison des raies entre 645 et 665nm')

# coordonnées des 2 points
X1 = None
X2 = None
continuum_fit = None

# fonctions gérant l'évenement 'button_click'
def firstclick(event):
    global fig, cid, X1
    # get the coordinate
    X1 = [event.xdata, event.ydata]
    fig.canvas.draw()
    fig.canvas.flush_events()
    # add a "+" on the position
    plt.plot(X1[0],X1[1],'k+',markersize=15)
    # Bind the button_press_event with the secondclick() method
    fig.canvas.mpl_disconnect(cid)
    cid = fig.canvas.mpl_connect('button_press_event', secondclick )
```

```
def secondclick(event):
    global fig, cid, X2, continuum_fit, wave_part5, masque
    # get the coordinate
    X2 = [event.xdata, event.ydata]
    # add a "+" on the position
    plt.plot(X2[0],X2[1],'k+',markersize=15)
    # compute and plot the fit
    f = interp1d([X1[0],X2[0]],[X1[1],X2[1]],␣
 ↪bounds_error=False,fill_value='extrapolate')
    continuum_fit = f(wave_part5)[masque]
    plt.plot(wave_part5[masque],continuum_fit,'k-',label='Continuum fit')
    plt.legend()
    fig.canvas.draw()
    fig.canvas.flush_events()
    # Empty the event connection
    fig.canvas.mpl_disconnect(cid)


# Bind the button_press_event with the firstclick() method. cid is the id of␣
 ↪the connection, usefull for deconnecting purpose
cid = fig.canvas.mpl_connect('button_press_event', firstclick )
```

Dans la figure ci-dessus, cliquez sur les deux extremités du continuum. Vous verrez alors apparaître une droite reliant ces deux points : il s'agit du fit de votre continuum.

Remarque : vous pouvez relancer la cellule au dessus de la figure pour recommencer la sélection avec deux nouveaux points et refaire votre fit en cas de fausse manip.

Nous avons maintenant un fit linéaire représentant de façon satisfaisante le continuum entre 645 et 665nm. **Complétez le code ci-dessous pour diviser chaque spectre par ce fit linéaire afin de ramener le continuum à 1 et affichez le résultat :**

[ ]:
```
# CODE A COMPLETER
```

Notre objectif est de trouver, pour chaque raie, si la forme de celle-ci se rapproche plus d'une gaussienne ou d'une lorentzienne, afin d'en déduire des propriétées physiques sur l'hydrogène dans ces étoiles. Pour cela, nous devons ramener notre continuum à 0 et ramener à 1 le maximum des raies en émission et à -1 le minimum des raies en absorption. Ceci nous permettra de comparer directement la forme des raies entre elles.

Pour ce faire, nous allons d'abord soustraire 1 à nos courbes afin de ramener le continuum à 0 :

[ ]:
```
# Division du spectre par le maximum/minimum de la raie :
Zeta_Tau_spectra = Zeta_Tau_spectra - 1
Tsih_spectra     = Tsih_spectra     - 1
Sirius_spectra   = Sirius_spectra   - 1
Alcyone_spectra  = Alcyone_spectra  - 1
Eta_Cas_spectra  = Eta_Cas_spectra  - 1
```

```
#########################################################

# Affichage
plt.figure()
plt.plot(wave_part5[masque],Zeta_Tau_spectra,label='Zeta_Tau')
plt.plot(wave_part5[masque],Sirius_spectra,label='Sirius')
plt.plot(wave_part5[masque],Alcyone_spectra,label='Alcyone')
plt.plot(wave_part5[masque],Eta_Cas_spectra,label='Eta_Cas')
plt.plot(wave_part5[masque],Tsih_spectra,label='Tsih')
plt.xlabel(f'$\lambda$')
plt.ylabel('Flux')
plt.legend()
```

Nous divisons ensuite chaque courbe par le maximum de sa valeur absolue, ce qui permet d'obtenir des raies dont le maximum est à 1 pour les raies en émission et à -1 pour les raies en absorption.

Complétez le code ci-dessous afin que chaque raie ait son maximum/minimum à 1 ou -1 et affichez le résultat :

[ ]:
```
# CODE A COMPLETER
```

Faites valider l'allure de vos spectres par un encadrant avant de passer à la suite !!

Question 14 : Comparez la forme des 2 raies en absorption. Ont-elles la même largeur à mi-hauteur ?

Répondre ici :

Question 15 : Le code ci-dessous va nous permettre d'ajuster chaque raie avec un profil gaussien et un profil lorentzien : executez la cellule ci-dessous sans la modifier.

[ ]:
```
# Définition du modèle gaussien :
def gauss(x, *p):
    A, mu, sigma = p
    return A*np.exp(-(x-mu)**2/(2.*sigma**2))

# Défintion du modèle Lorentzien :
def lorentzian(x, *p):
    A, mu, gam = p
    return (A/np.pi) * (gam/2) / ( (gam/2)**2 + ( x - mu )**2)


class do_fit:

    def __init__(self,xdata,ydata,name):
        self.xdata = np.copy(xdata)
        self.ydata = np.copy(ydata)
        self.name  = name
```

```
        # Affichage du spectre
        fig = plt.figure()

        plt.plot(xdata,ydata,label=name)
        plt.xlabel(f'$\lambda$ [nm]')
        plt.ylabel('Flux')
        plt.legend()

        self.titles = [
        'Cliquez sur le continuum : gauche',
        'Cliquez sur le continuum : droite',
        'Cliquez sur la raie à mi-hauteur : gauche',
        'Cliquez sur la raie à mi-hauteur : droite',
        'Cliquez sur le max/min de la raie',
        name]

        # coordonnées des 5 points
        self.X = []
        self.k = 0
        plt.title(self.titles[self.k])

        # Bind the button_press_event with the firstclick() method. cid is the␣
↪id of the connection, usefull for deconnecting purpose
        self.cid = fig.canvas.mpl_connect('button_press_event', self.click )

    # fonctions gérant l'évenement 'button_click'
    def click(self,event):
        # get the coordinate
        self.k+=1
        plt.title(self.titles[self.k])
        self.X.append([event.xdata, event.ydata])
        fig.canvas.draw()
        fig.canvas.flush_events()
        # add a "+" on the position
        plt.plot(self.X[-1][0],self.X[-1][1],'k+',markersize=15)

        # Disconnect event and plot the Gaussian fit when 5 points have been␣
↪selected
        if len(self.X)>=5:
            fig.canvas.mpl_disconnect(self.cid)
            # re-normalise continuum of the specific line
            f = interp1d([self.X[0][0],self.X[1][0]],[self.X[0][1],self.
↪X[1][1]], bounds_error=False,fill_value='extrapolate')
            continuum_fit = f(self.xdata)+1

            self.ydata = ((self.ydata+1) / continuum_fit) - 1
            # compute sigma using FWHM
```

```
            self.fwhm = np.abs(self.X[2][0] - self.X[3][0])
            self.sigma = self.fwhm / 2.355
            # get the mu value (center of gaussian)
            self.mu = self.X[4][0]
            # get the amplitude
            self.A  = self.X[4][1]

            # replot figure
            plt.clf()
            plt.plot(self.xdata,self.ydata,label=self.name)

            # use scipy for gaussian fit
            self.coeff_G, self.var_matrix_G = curve_fit(gauss, self.xdata, self.
↪ydata, p0=[self.A,self.mu,self.sigma])
            # plot the gaussian fit
            plt.plot(self.xdata,gauss(self.xdata,*self.
↪coeff_G),'r--',label=f'Gaussian fit : FWHM={self.coeff_G[-1]*2.355:.2f}')

            # use scipy for lorentz fit
            self.coeff_L, self.var_matrix_L = curve_fit(lorentzian, self.xdata,␣
↪self.ydata, p0=[self.A,self.mu,self.fwhm])
            # plot the gaussian fit
            plt.plot(self.xdata,lorentzian(self.xdata,*self.
↪coeff_L),'g--',label=f'Lorentzian fit : FWHM = {self.coeff_L[-1]:.2f}')

            plt.legend()
            fig.canvas.draw()
            fig.canvas.flush_events()
```

Grâce à la cellule ci-dessus, nous avons maintenant une fonction "do_fit()" permettant d'ajuster une Gaussienne et une Lorentzienne sur un spectre. Pour cela, il suffit d'appeler la fonction avec comme argument le spectre ainsi que le nom de l'étoile. En lançant la fonction, on obtient une figure sur laquelle il faut cliquer en 5 points dans l'ordre suivant :

- 1er point : sur le continuum à gauche
- 2nd point : sur le continuum à droite
- 3e point : sur la raie, à mi-hauteur, à gauche
- 4e point : sur la raie, à mi-hauteur, à droite
- 5e point : sur le maximum/minimum de la raie en émission/absorption

Pour chacune des cellules ci-dessous : executez la cellule, et cliquez sur la figure afin d'indiquer à la fonction les 5 points à prendre en compte pour le fit. Veillez à bien respecter l'ordre de sélection de ces 5 points !

[ ]:
```
Sirius_fit = do_fit(wave_part5[masque],Sirius_spectra,"Sirius")
```

[ ]:
```
Eta_Cas_fit = do_fit(wave_part5[masque],Eta_Cas_spectra,"Eta Cas")
```

Ces deux raies en absorption ont elles un profil de type Lorentzien ? Gaussien ? Quelle est la largeur à mi-hauteur estimée par la fonction (FWHM, dans la légende de la figure) ?

Répondre ici :

Question 16 : Le profil Lorentzien d'une raie provient des effets de collisions des atomes dans l'atmosphère de l'étoile. Le profil Lorentzien s'élargit à mesure qu'il y a plus de collisions. Ces collisions sont générées par une augmentation de la température. Cela est-il cohérent avec ce que vous observez ?

Répondre ici :

1.6 6 - Les étoiles de type Be ou étoile B à raie d'émission

Les étoiles Be sont des étoiles de type B - donc chaudes (température de 10000 à 30000K) dont le spectre a montré au moins une fois une raie en émission, généralement une raie de Balmer. Ces raies en émission peuvent se présenter sous différentes formes. Elles proviennent d'un disque équatorial dont l'émission s'ajoute au spectre d'absorption de la photosphère de l'étoile. L'étoile centrale de type B émet notamment dans l'Ultra-Violet et ionise ce disque qui ré- émet l'énergie à de plus grandes longueurs d'onde comme dans le domaine visible. La même étoile Be peut avoir un profil spectral variable selon l'angle d'observation de ce disque (Figure 6). Ces émissions peuvent être accompagnées d'un décalage doppler des raies d'émission vers le bleu pour la partie du disque qui se rapproche de nous et vers le rouge pour la partie qui s'éloigne de nous (cf Figure 7).

Figure 6 - Modèle d'une étoile Be classique montrant le type de spectre observé en fonction de l'orientation selon laquelle l'observateur voit le disque d'émission (Kogure & Hirata, 1982).

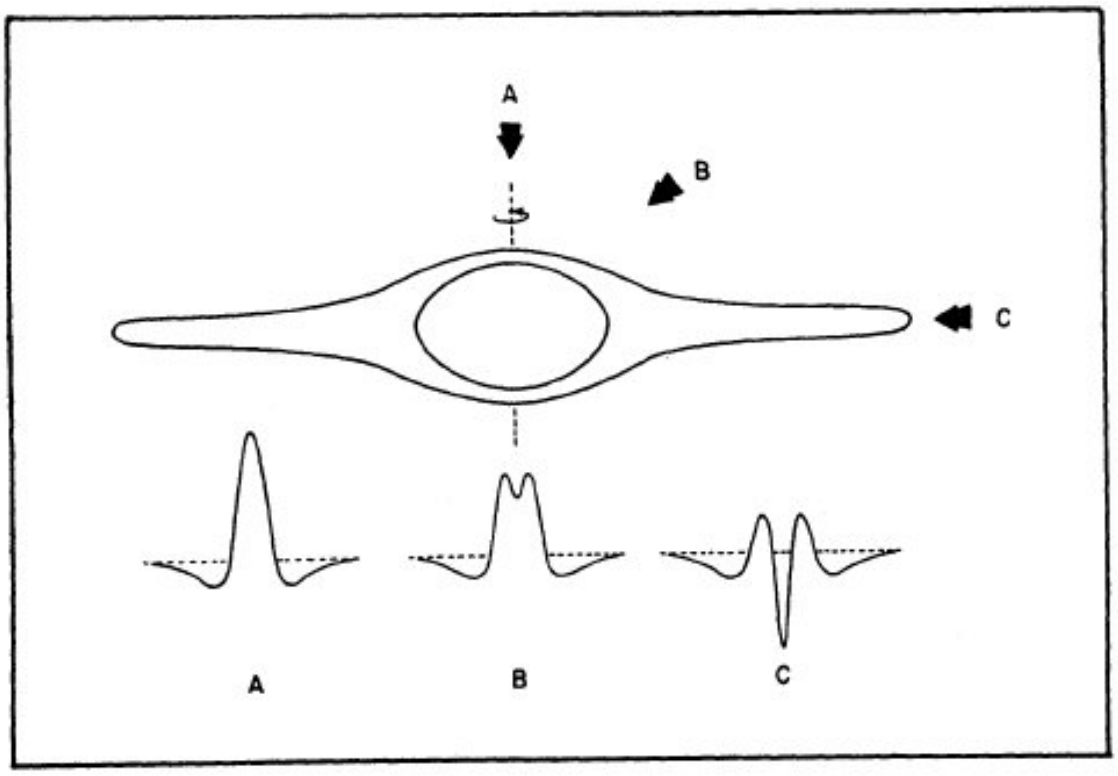
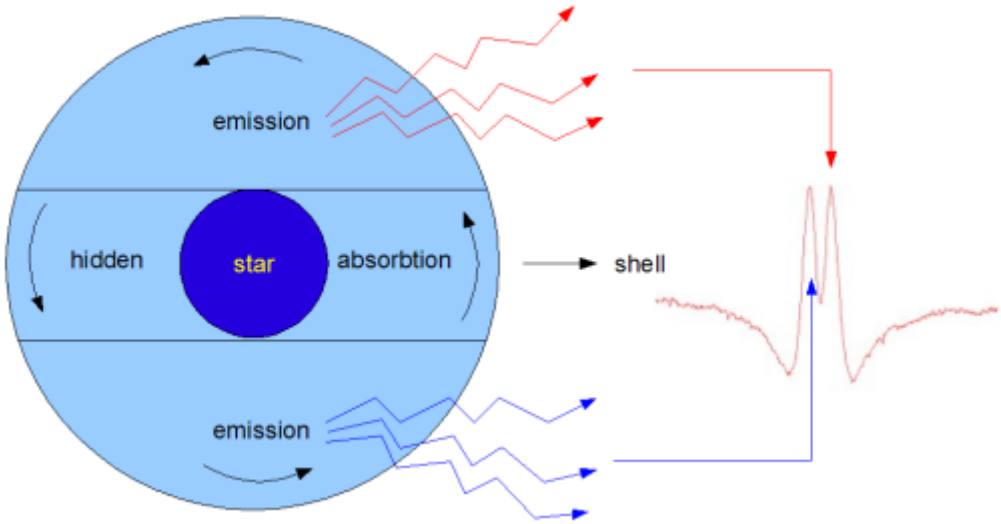


Figure 7 - Modèle d'une étoile Be classique montrant le décalage des raies d'émission de chaque partie du disque observé.



Question 17 : Quelles sont les trois étoiles parmi Tsih, Zeta Tau, Sirius, Eta Cassiopée et Alcyone qui correspondent à la définition d'une étoile Be ? Vu la forme des raies et en vous aidant de la Figure 6, pouvez-vous estimer l'orientation dans laquelle nous voyons chacune de ces trois étoiles ?

Répondre ici :

Question 18 : L'une de ces étoiles montre clairement une raie déformée et assez élargie. En comparant la largeur à mi-hauteur de cette raie avec la largeur à mi-hauteur de la raie la plus fine, estimez l'élargissement de la raie en nm. Vous pourrez vous aider de la fonction "do_fit" pour mesurer plus précisément la largeur à mi-hauteur de chaque raie.

[ ]:
```
# Code à compléter
```

[ ]:
```
# Code à compléter
```

Répondre ici :

Question 19 : En vous rappelant que le décalage Doppler (pour des vitesses v beaucoup plus petites que la vitesse de la lumière c) s'écrit = v/c , déduisez la vitesse maximale sur les bords du disque d'émission.

Répondre ici :

1.6.1 Conclusion :

A compléter
