

Spectro-Python

Auswertung optischer astronomischer Spektren mit Hilfe von Python 3

Dr. Lothar Schanne

February 8, 2022

Contents

I. Motivation	4
II. Installation einer persönlichen Python-Version	4
1. Allgemeines	4
1.1. Anmerkung:	5
2. Installation einer Python-Version im eigenen User-Verzeichnis	5
3. Installation einer Entwicklungsumgebung (Spyder) und SciPy (wissenschaftliches Rechnen)	6
4. Verwendung von Spyder	7
5. Spektroskopiespezifische Python-Bibliotheken	7
6. Zur Einführung ein erstes spektroskopisches Programm	8
III. Manipulation von 1d-fits-Dateien (1d-Spektren)	11
7. Badpixel-Filter	11
8. Kontrolle und Korrektur des Headers einer Serie von 1d-Spektren im fits-Format	13
9. Erzeugen eines definierten Wellenlängenausschnitts aus einem fits-1d-Spektrum	14
10. Zeitreihen von 1d-Spektren im fits-Format auf einen Wellenlängenbereich beschneiden und grafisch darstellen	14
11. Übersetzen eines 1d-Spektrums im fits-Format in ascii-Dateien	16
12. Erzeugen eines gemeinsamen Wellenlängenausschnitts einer Serie von 1d-Spektren im fits-Format und abspeichern als fits- und als ascii-Dateien	16
13. Rebinnen einer Serie von 1d-Spektren im fits-Format auf eine gemeinsame Schrittweite und einen gemeinsamen Wellenlängenbereich sowie Bildung eines gemittelten Spektrums	19

IV. Umgang mit 1d-Spektren im ASCII-Format	20
14. Plotten eines ascii-1d-Spektrums	20
15. Plotten und erzeugen eines Ausschnitts aus einem ascii-1d-Spektrum	20
16. Erzeugen von Differenzspektren	23
V. Theoretische Spektren und Konvolution (Faltung) von Modell-1d-Spektren mit Gaussfunktionen	23
17. Konvolvieren eines 1d-fits-Spektrums auf eine bestimmte Auflösung R	23
18. Modifizierung eines 1d-Spektrums im ascii-Format (convol_dat.py)	23
19. Modifizierung eines theoretischen Spektrums aus der Pollux-Datenbank auf eine bestimmte Schrittweite und Auflösung	27
20. Bezug eines theoretischen Spektrums aus einer Datenbank von PyAstronomy mit einer bestimmten Auflösung R	30
21. Vergleich von theoretischem und gemessenen Spektrum (overplot)	30
VI. Normierung von 1d-Spektren	33
22. Interaktive Festlegung von Normierungsstützstellen und Normierung mittels eines Splines	33
23. Normierung einer Serie von 1d-Spektren mittels einer Liste festgelegter Wellenlängen der Normierungsstützstellen	33
24. Allgemeine Routine zur Normierung ganzer Spektrenserien im fits-Format	34
VII. Bestimmung des Signal-Rausch-Verhältnisses	42
VIII. Baryzentrische Korrektur (barycentric correction BC)	42
25. Berechnung der BC	42
26. BC-Korrektur einer Serie von 1d-Spektren im fits-Format durch Dopplerverschiebung	44
IX. Bestimmung von Radialgeschwindigkeiten	44
27. Interaktive Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im ascii-Format (tab-separiert)	44
28. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format	47
28.1. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels einer Regression	49
28.2. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels einer RBF	51

28.3. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels eines kubischen Splines	51
28.4. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels eines Gauß-fittings	51
X. Bestimmung von Äquivalentweiten	53
29. Bestimmung der Äquivalentweite einer Linie in einer Spektrenserie im fits-Format	53
XI. Kreuzkorrelationen (KK)	53
30. KK für 1d-Spektren des Dateityps fit	53
31. KK für 1d-Spektren im ASCII-Format	57
32. KK einer Spektrenserie im fits-Format mit einem Template	57
33. KK einer Spektrenserie im ascii-Format mit einem Template	57

Part I.

Motivation

Die Auswertung von CCD-Aufnahmen mit optischen astronomischen Spektren von Sternen, Emissionsnebeln etc. wurde/wird im professionellen Bereich überwiegend mit den öffentlich zugänglichen und kostenfreien Programmpaketen ESO-Midas¹ und IRAF² sowie institutseigenen Programmen durchgeführt.

Hobbyastronomen arbeiten großenteils lieber mit Programmen für die Windows-Betriebssysteme, die meist intuitiver und leichter zu erlernen, andererseits aber auch bzgl. ihrer Fähigkeiten begrenzt sind und zudem Blackboxes darstellen (der Nutzer kann nicht einsehen, was das Programm im einzelnen macht). Als oft verwendete Programme seien genannt IRIS³ und VSpec⁴, RSpec⁵ und BASS⁶.

Der Autor verwendet auf einer Linux-Plattform (Ubuntu 20.04) üblicherweise MIDAS. Damit lässt sich praktisch alles erledigen, was die Spektroskopikerpraxis erfordert, allerdings ist auch eine erhebliche Einarbeitungsanstrengung zu leisten.

Häufig ergibt sich aber auch der Wunsch nach Spezialauswertungen, die vermutlich auch in MIDAS oder IRAF möglich sind, aber das Wissen und Können des Anwenders übersteigen. Da ist es u.U. vorteilhaft, auf universell einsetzbare Programmiersprachen zurück zu greifen. Hier die Interpretersprache *Python*.

Python hat gegenüber anderen Hochsprachen wie C++ den Vorteil einer einfacheren Lesbarkeit von Programmen (Skripten) und das schnellere Erlernen der Sprache. Außerdem gibt es eine Unzahl von wissenschaftlichen Modulen im Internet, die sehr einfach in die eigenen Programme integriert werden können („*import*“). Wir möchten jetzt nicht hier in das Für und Wider der einzelnen Programmiersprachen eintreten. Dazu gibt es genügend gute Literatur, sowohl in Form von Büchern wie auch von Internetseiten. Der Autor verwendete für den Einstieg in Python ein Buch von Bernd Klein, *Einführung in Python* in der 3. Auflage⁷, als Nachschlagewerk den Klassiker von Mark Lutz⁸ und für wissenschaftliches Rechnen (scipy, numpy, pandas, matplotlib) das Buch *Data Science mit Python*⁹ sowie *Datenanalyse mit Python*.¹⁰

Part II.

Installation einer persönlichen Python-Version

1. Allgemeines

Python gibt es in unterschiedlichen Distributionen und Versionen¹¹. Beispielsweise ist die veraltete, aber immer noch funktionierende Version Python 2 derzeit im letzten Release 2.7 vorhanden. Die

¹<http://www.eso.org/sci/software/esomidas/>

²<http://iraf.noao.edu/>

³<http://www.astrosurf.com/buil/iris-software.html>

⁴<http://www.astrosurf.com/vdesnoux/>

⁵<https://www.rspect-astro.com/>

⁶http://aeselas.com/mediapool/142/1423849/data/DOCUMENTOS/BASS_Project_1_.pdf

⁷Hanser Verlag, 2018.

⁸Learning Python, 5th Edition Powerful Object-Oriented Programming

By Mark Lutz

Publisher: O'Reilly Media

Release Date: June 2013

Pages: 1648

⁹Autor: VanderPlas, Jake von mitp Verlag 549 Seiten, Softcover ersch. 01/2018 ISBN: 978-3-95845-695-2

¹⁰Wes McKinney, O'Reilly-Verlag, 2. Auflage 2019, 522 Seiten, Softcover, ISBN 978-3-96009-080-9

¹¹<https://www.python.org/>

neueren Python-Distributionen sind in Python 3 geschrieben, die aktuelle stabile Version ist derzeit Python 3.9.

In LINUX sind Teile des Betriebssystems in Python geschrieben, weshalb bei der Einrichtung des jeweiligen LINUX-Systems automatisch auch Python 2 und Python 3 mit installiert werden, und zwar im nur mit Administratorrechten zugänglichen Dateisystem. In Ubuntu 16.04 ist es beispielsweise Python 3.5. Verändert man diese Python-Bibliotheken, indem über einen typischen Befehl mit Administratorrechten wie „*sudo apt-get install 'Pythonmodul'*“ ein neuer Modul eingespielt wird, kann es passieren, dass vom Betriebssystem benötigte Hintergrund-Dateien mit anderen Versionen überschrieben werden und anschließend dadurch unbeabsichtigte Probleme auftauchen. Dem Autor ist es passiert, dass nach einer solchen „Aktualisierung“ der Bildschirm nach dem PC-Neustart schwarz blieb, weil das Grafiksystem (X-Server) nicht mehr *seine* Bibliotheken fand. Glücklicherweise gibt es dann in LINUX noch die Konsolen, die ohne das Grafiksystem auskommen und mit denen die Fehler repariert oder rückgängig gemacht werden können.

Aus diesem Grund empfehlen wir jedem, der frei in Python programmieren möchte, eine eigene Pythonversion ohne Administratorrechte in seinem Userverzeichnis einzurichten. Und dann später für eigene Arbeiten immer dieses Python aufzurufen. So kommen das Betriebssystem-Python und das eigene nie in Konflikt.

1.1. Anmerkung:

Es gibt eine Möglichkeit, in einem Schritt eine sehr umfangreiche Python3-Distribution für das wissenschaftliche Arbeiten zu installieren: Anaconda¹². Die kostenlose Version enthält über 1400 data science packages und verwaltet alle Hintergrundbibliotheken und -skripte automatisch mit dem eigenen Verwaltungsprogramm *conda*. Wir empfehlen jedem Python-Anfänger diese Distribution, auch wegen der leichten Instaliierbarkeit. Das Nachinstallieren von Bibliotheken für wissenschaftliche Zwecke, wie in den Abschnitten des Abschnitts 3 beschrieben, entfällt in diesem Falle, da sie bereits in Anaconda vorinstalliert sind. Einige Pakete, die in der Spektroskopie angewendet werden können (wie PyAstronomy), fehlen, lassen sich aber manuell mit *pip* nachinstallieren.

2. Installation einer Python-Version im eigenen User-Verzeichnis

Falls man die Anaconda-Distribution nicht möchte lädt man sich von der Internetseite der python.org¹³ die gewünschte Python-Version für die eigene Plattform zum Download aus. Als Beispiel nehme ich für Linux das komprimierte Archiv *Python-3.7.0.tar.xz*. In diesem Archiv gibt es die Datei README.rst, die auf jeden Fall zuerst gelesen werden muß! Nützlich sind auch die Informationen auf der Seite <https://docs.python.org/3/>. Insbesondere das Kapitel Python Setup and Usage¹⁴. Hier findet man Installationsanleitungen für die unterschiedlichen Betriebssysteme (Linux, Windows, Macintosh). Man entpackt das Archiv in dem eigenen Python-Ordner und installiert/übersetzt dann manuell (in LINUX) in einer Konsole im eigenen Python-Ordner mit der üblichen Befehlsreihenfolge *./configure --prefix=Zielverzeichnis --enable-optimizations, make* und *make altinstall*. Aber Vorsicht, um alle Funktionalitäten von Python 3.7 zu erhalten benötigt man eine Reihe von Entwicklerbibliotheken, die auf dem PC bereits vorhanden sein müssen. Für Ubuntu 18.04 bis 21.04 ist der ganze Installationsvorgang vollständig auf einer Webpage¹⁵ beschrieben. Bitte sorgfältig beachten!

Damit die persönliche Python-Version mit dem Befehl *python3.X* in einer Konsole auch gefunden wird sollte folgender Eintrag in die Datei *.profile* im Home-Verzeichnis des Users eingetragen werden, am Besten am Ende:

```
PATH=$PATH:$HOME/python3.7/bin
```

Die geänderte Datei abspeichern, als User sich abmelden und wieder neu anmelden (dann wird die Datei *.profile* neu eingelesen). Dann sollte in einer Konsole das Aufrufen von Python mit *python3.X*

¹²<https://www.anaconda.com/>

¹³<https://www.python.org/>

¹⁴<https://docs.python.org/3/using/index.html>

¹⁵https://wiki.ubuntuusers.de/Python/manuelle_Installation/

(X durch die Versionsnummer ersetzen) funktionieren. Der erfolgreiche Start von Python wird in der Konsole erkenntlich durch den Ausdruck von

```
lothar@tux: ~$ python3.7
Python 3.7.0 (default, Aug 13 2018, 17:47:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Nach dem neuen Prompt `>>>` können jetzt die Pythonbefehle für interaktives arbeiten eingegeben werden¹⁶.

Wenn alle diese Tipps berücksichtigt wurden hat man innerhalb einiger Minuten bis einiger Stunden¹⁷ eine eigene (userspezifische) Python-Grundversion installiert, welche die Version des Betriebssystems nicht tangiert oder stört und die jederzeit verändert, aktualisiert und ergänzt werden kann. Auf die gleiche Weise kann man sich mehrere unterschiedliche Python-Versionen parallel in unterschiedlichen Verzeichnissen in das eigene User-Verzeichnis installieren, auch die Anaconda-Distribution.

Das so installierte Python enthält nur die Grundversion des installierten Release. Für wissenschaftliches Arbeiten, insbesondere für das Arbeiten mit numerischen Daten wie Spektren, benötigen wir noch eine Reihe von Zusatzmodulen.

3. Installation einer Entwicklungsumgebung (Spyder) und SciPy (wissenschaftliches Rechnen)

Die Entwicklung von Programmen (Skripten) geht am bequemsten mit einer Entwicklungsumgebung (englische Abkürzung IDE) wie IDLE (in Python 3 integriert) oder *Spyder*. Letztere bevorzugt der Autor. Um Spyder zur Verfügung zu haben sollte ein Zusatzmodul installiert werden namens *Jupyter*. Es ist Bestandteil eines großen Pakets von wissenschaftlichen Zusatzbibliotheken, die von der community der SciPy.org entwickelt werden¹⁸. Am besten installieren wir alle die 6 open-source Pakete namens *NumPy* (numerisches Rechnen), *SciPy* library (fundamentale Bibliothek), *Matplotlib* (Grafiken), *IPython* (verbesserte Python-Konsole mit vielen praktischen Funktionalitäten, die in der normalen Pythonkonsole nicht enthalten sind), *Sympy* (für symbolisches Rechnen) und *pandas* (Verarbeitung tabellierter Daten vom Feinsten, Datenstrukturen und -analyse). Im Paket IPython ist die oben erwähnte Entwicklungsumgebung *Spyder3* enthalten und wird automatisch mit installiert.

```
Jupyter wird unter Verwendung des Paketverwaltungsprogramms pip mit der Befehlsfolge
pip3 install --upgrade pip (um pip3 in aktueller Form zu haben)
pip3 install jupyter
```

Die anderen analog. Anschließend kann man von einer Konsole aus mit dem Befehl *spyder3* die Entwicklungsumgebung aufrufen und damit arbeiten. Alternativ auch mit den für das Betriebssystem üblichen Programmstart Routinen.

Das Paketverwaltungsprogramm *pip* muss konsequent angewendet werden. Entweder alles mit *pip* installieren oder nichts, sonst kann es Durcheinander mit den Hintergrundskripten geben. Hat man beispielsweise die Anaconda-Distribution installiert, sollte man nicht deren Paketverwaltungsprogramm *conda* anwenden und dann zwischendurch mal *pip*. Dann immer *conda!* Allerdings kann *conda* nur auf Pakete zugreifen, die in der Anaconda-Distribution integriert sind. Dies garantiert, dass es keine Abhängigkeitsprobleme von Bibliotheken gibt und das Python3-System in sich konsistent bleibt. Mit dem Nachteil, dass man auf die Anaconda-Distribution beschränkt bleibt¹⁹.

In den späteren Abschnitten werden wir sehen, welche zusätzlichen Module wir noch installieren werden, also solche, die für Astroanwendungen essentiell sind.

¹⁶Man verlässt Python mit dem Befehl `quit()`.

¹⁷Die Quellcodes werden alle kompiliert, was je nach Schnelligkeit des eigenen PC/Laptops sehr unterschiedlich lange dauert.

¹⁸<https://scipy.org/>

¹⁹Man kann allerdings andere Pakete mit *pip* installieren. Das sollte man aber nur tun, wenn die Bibliothek nicht mit *conda* installiert werden kann.

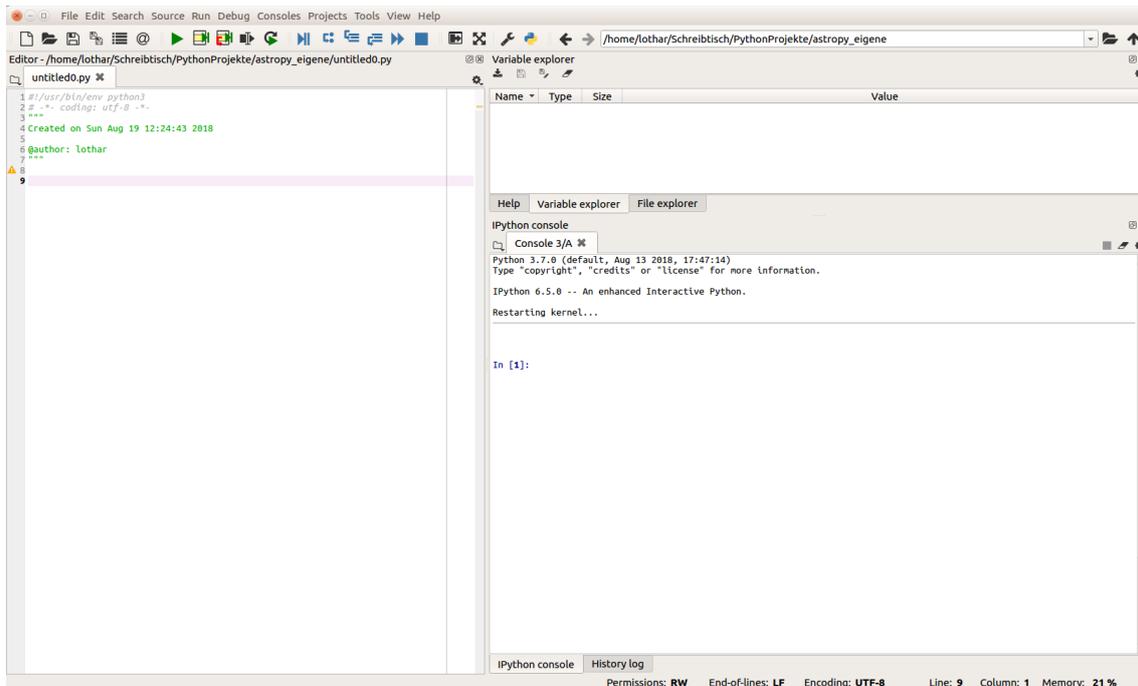


Figure 4.1: Spyder-Fenster nach dem Start.

Wichtig ist jetzt sich mit der Sprache Python etwas zu beschäftigen, falls man blutiger Anfänger in Python ist. Dazu muss man schon einige Tage investieren um die wichtigsten Grundlagen wie Datenstrukturen, Programmanweisungen wie Schleifen, Datenmanipulationen etc. zu erlernen und die Arbeitsweise in einer Python-Konsole zu üben. Die Lernkurve ist aber sehr steil!

4. Verwendung von Spyder

Wie sieht jetzt diese Entwicklungsumgebung Spyder aus und wie arbeitet man damit?

Abb. 4.1 zeigt das Fenster, das sich nach dem Start von Spyder3 öffnet. Rechts unten ist die *IPython-Konsole*, in der man Python-Befehle unmittelbar ausführen kann. Rechts oben ist der *Variable explorer* geöffnet, der bereits definierte Variablen anzeigt. Und links ist das Fenster des Editors geöffnet. Darin werden die Programme (Skripte) entworfen. Sie können dann mit F5 oder dem grünen Pfeil-button in der Konsole rechts unten ausgeführt werden. Die Ergebnisse werden unmittelbar in der Python-Konsole angezeigt. Bzgl. der detaillierten Beschreibung der Arbeitsweise mit Spyder verweise ich auf die Hilfe (Help) im Menü oben oder die Spyder-Dokumentation. Spyder unterstützt auch Deutsch.

Für ein reibungsloses Arbeiten mit Spyder, insbesondere bzgl. der Anzeige von aktiv bearbeitbaren Grafiken, empfehle ich die Einstellungen für die IPython-Konsole nach Abb. 4.2.

5. Spektroskopiespezifische Python-Bibliotheken

Nach der Installation der Pythondistribution von Anaconda sind bereits astronomiespezifische Pythonpakete installiert, insbesondere das umfangreiche *Astropy*. Außerdem gibt es eine Menge von Modulen, die in der Webpage <https://www.astropy.org/affiliated/> zu finden sind und über *conda* oder den *anaconda-navigator* geladen werden können. Darunter für die Spektroskopie wertvolle Module wie *specutils*, *linetools*, *pyspeckit*, *spectral-cube*, *SpectraPy*. Neben diesen an Astropy gelehnten Paketen gibt es noch weitere spektroskopiespezifische, die mit *pypi* installiert werden müssen. Praktisch in der Verwendung ist vor allem *PyAstronomy* (<https://pyastronomy.readthedocs.io/en/latest/>).

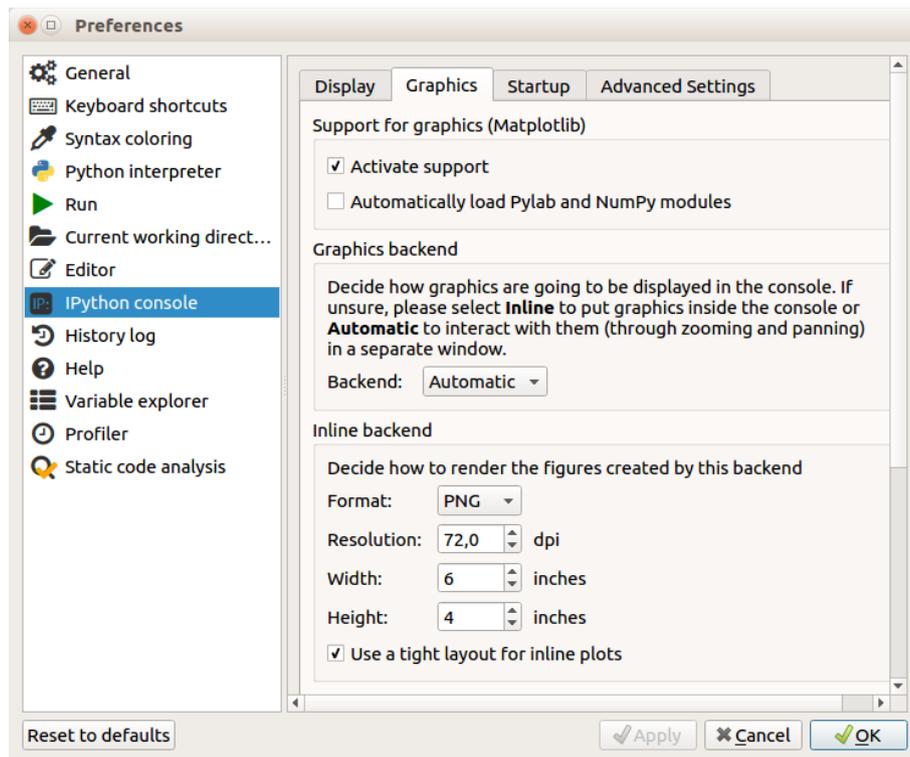


Figure 4.2: Einstellungen für die IPython-Konsole in Spyder.

6. Zur Einführung ein erstes spektroskopisches Programm

Als erste Aufgabe wollen wir uns ein bereits reduziertes 1d-Spektrum im üblichen fits-Format grafisch anschauen. Dazu müssen wir uns ein Skript schreiben, das die notwendigen Anweisungen für Python3 enthält. Wir setzen hier voraus, dass der Anwender die wichtigsten Python-Befehle kennt, also den Umgang mit Dateien und Variablen, Bildung von Schleifen und darin eingebettete Anweisungen, also minimale Programmierkenntnisse mit Python3. Und dass die zu importierenden Module auch installiert sind. Wir öffnen das Skript mit der IDE *Spyder*.

In Abb. 6.1 sehen wir das Skript *1d_fitSpektrum_ansehen.py*, in einem Texteditor geöffnet. Dies ist ein einfacher Textfile, der in jedem Editor angezeigt und manipuliert werden kann. Dass er ein Pythonprogramm ist drücken wir mit der Extension *.py* im Dateinamen aus. Sehen wir uns diesen Text an. In rot ist ein beschreibender Text in dreifachem " eingeschlossen. Das ist der Text der beim Aufrufen der Hilfe zu der Datei angezeigt wird. Er sollte möglichst informativ sein.

Darunter kommen drei *import*-Befehle. Mit der Importierung eines Moduls werden alle darin enthaltenen Klassen und Funktionen im laufenden Programm verfügbar. Natürlich können nur Module importiert werden, die in unserem Pythonsystem auch bereits installiert wurden (mit *conda* oder *pip*). Hier werden die Module (= Programmpakete) aufgerufen, die folgendes bewirken:

- *numpy* ist ein Modul zur schnelleren Verarbeitung numerischer Daten²⁰. Es stellt insbesondere Arrays zu Verfügung, in denen Daten gleichen Typs gespeichert und anschließend schnell umgruppiert, geändert und verarbeitet werden können. *numpy* erhält die Abkürzung *np* und wird unter diesem *Alias (Spitznamen)* im Skript verwendet.
- *astropy.io* ist ein Bestandteil des astronomischen Pakets *Astropy*²¹. Es dient zum Ein- und Auslesen von Datenfiles, z.B. fits-Files. Hier wird nur das Paket *fits* aus *astropy.io* importiert, weil wir nicht mehr brauchen.
- Das Modul *matplotlib* enthält eine wahre Fundgrube von grafischen Möglichkeiten²². Auch

²⁰<http://www.numpy.org/>

²¹<http://www.astropy.org/>

²²<https://matplotlib.org/>

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 1d Spektrum ansehen.py
5 #Ansehen eines 1d-Spektrums, Auslesen und Anzeige
6 der wichtigsten Headerdaten.
7 Plotten des Spektrums.
8
9 Created on Sunday Jul 22 2018
10 @author: lothar
11 """
12
13 import numpy as np
14 from astropy.io import fits
15 import matplotlib.pyplot as plt
16
17 # Pfad und Name des Spektrumfiles bitte anpassen
18 file = input("Pfad und Filebezeichnung eingeben: ")
19
20 # Lesen des Spektrums
21 sp = fits.open(file)
22
23 # Header lesen und in der Konsole ausdrucken
24 HD = dict(sp[0].header)
25 print("\n\nHeader des Spektrums :\n")
26 print(HD)
27
28
29 # Erzeugen von Arrays mit den Wellenlängen und Fluxes des Spektrums
30 flux = np.array(sp[0].data)
31 wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
32 for i in range(sp[0].header["NAXIS1"]):
33     wave[i] = (
34         sp[0].header["CRVAL1"]
35         + (i + 1 - sp[0].header["CRPIX1"]) * sp[0].header["CDELTA1"]
36     )
37 # In der Liste wave sind die Wellenlängen der Pixel enthalten
38 # In der Liste flux die entsprechenden Intensitäten
39
40 # Schliessen des fits-file
41 sp.close()
42
43 # Plot gesamtes Spektrum
44 fig = plt.figure(1, figsize=(14, 10))
45 plt.plot(wave, flux, "-", linewidth=0.2)
46 plt.xlabel("Wellenlänge [Angström]", fontsize=8)
47 plt.ylabel("ADU", fontsize=8)
48 plt.title("Spektrum " + file, fontsize=10)
49 plt.xticks(fontsize=8)
50 plt.yticks(fontsize=8)
51 plt.grid(True)
52
53 # Zeigen des Plots, wenn das Skript in einer normalen Python-Konsole
54 # durchgeführt wird. Für die IPython-Konsole in Spyder ist das nicht nötig.
55 plt.show()

```

Figure 6.1: Programmcode zur Ansicht eines 1d-Spektrums im fit-Format und der Header-Angaben.

es muss installiert sein, bevor wir dann das Teilmodul *matplotlib.pyplot* unter dem Alias *plt* importieren können.

Damit haben wir alles zur Verfügung um das eigentliche Programm zu beginnen.

Natürlich müssen wir dem Programm mitteilen, welches Spektrum im fits-Format wir uns ansehen wollen. Das geschieht mit der Anweisung

```
file = input('Pfad und Filebezeichnung eingeben: ')
```

Wir definieren eine Variable namens *file*, der wir den String (=Zeichenfolge) zuordnen, den wir in der Konsole nach der Aufforderung 'Pfad und Filebezeichnung eingeben:' manuell eingeben. Hier muss der komplette relative oder absolute Pfad zur fits-Datei eingegeben werden, wenn sie nicht im aktuellen Arbeitsverzeichnis steht (das rechts oben im Spyderfenster gewählt und angezeigt wird).

In Zeile 24 leisten wir uns einen Luxus: Wir wollen wissen, was im Header des Spektrums steht, also um welches Objekt es sich handelt, wann die Aufnahme geschah etc²³. Dazu müssen wir zuerst das File öffnen (Zeile 21). Hier sehen wir die typische Formulierung einer objektorientierten Sprache. Wir rufen aus der Instanz *fits* (die wir ja in Zeile 14 importiert haben) die Methode *fits.open()* auf, die dann das vorher in *file* definierte File öffnet (lesbar macht). Diesem geöffneten Objekt geben wir einen beliebigen Bezeichner, hier *sp*.

Das 1d-Spektrum enthält nur eine Dimension (entlang der Dispersion), diese wird durch *sp[0]* zugänglich. Und da wir den header sehen wollen gibt die Methode *sp[0].header* genau den kompletten Inhalt des Headers zurück, der nach der manuell programmierten Erläuterung 'Header des Spektrums :' in der Konsole ausgedruckt wird. Um den Header in kompakter Form anzuzeigen wurde er vorher noch in ein Dictionary-Objekt verwandelt (Zeile 24). Lassen wir das Programm aus dem Editor von Spyder heraus in der Konsole laufen (Taste F5 oder grüner Pfeil), ergibt sich der Headerausdruck wie in Abb. 6.2. Alle Keywords des Headers sind in der Konsole rechts unten ausgegeben. Und auch alle im Programmablauf definierten Variablen werden im Variablenexplorer aufgeführt (Fenster rechts oben), inkl. Variablentyp, -größe und die ersten Werte dazu. Wichtig für die weiteren Berechnungen sind die Headerwerte:

²³Datenfiles im Format fits bestehen zumindest aus einem Header und einem Datenteil. In dem in Zeile 21 erzeugten Objekt *sp* gibt es unter dem Index 0 (*sp[0]*) den Header (Zugriff mit dem Ausdruck *sp[0].header*) und die Datentabelle (Intensitäten (Flux) der Pixel, Zugriff mit *sp[0].data*).

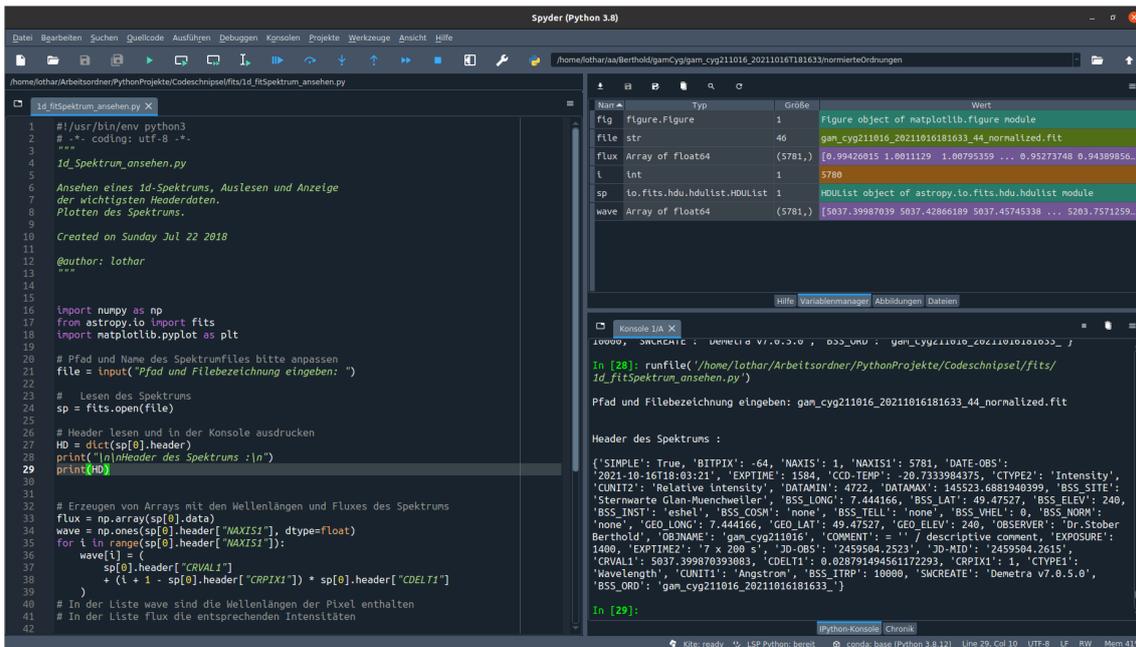


Figure 6.2: Ausdruck des Headers in der Konsole nach Start des Programms aus dem Editor von Spyder heraus.

1. $NAXIS1 = 5781$ / Axis length = Anzahl der Pixel/Wellenlängenpunkte
2. $CRVAL1 = 5037.399870393083$ = Start der Wellenlängenskala in Angström
3. $CDELTA1 = 0.028791494561172293$ = Schrittweite, also das Wellenlängenintervall der einzelnen Pixel
4. $CRPIX1 = 1$ = Referenzpixel, auf das sich $CRVAL1$ bezieht.

Mit den vier Größen lässt sich jedem Pixel die Wellenlänge zuordnen.²⁴

Im weiteren Verlauf des Programmtextes beginnen wir in Zeile 29 mit der Berechnung der Wellenlängen, die zu den Fluxwerten gehören, damit wir sie anschließend grafisch als Spektrum darstellen können.

Wir erinnern uns: In $sp[0].data$ stehen die 5781 Fluxwerte. Diese übergeben wir in Zeile 30 in ein numpy-Objekt, nämlich ein numpyarray (eine spezielles Array für Daten gleichen Typs) und nennen diese Variable $flux$. In Zeile 31 erzeugen wir ein numpyarray namens $wave$ mit lauter Einsen (1.), und zwar mit genau $NAXIS1$ (hier 5781) Stück. Dieses füllen wir im weiteren Verlauf mit den Wellenlängen der Pixel auf. Dazu berechnen wir in der Schleife in Zeilen 32 bis 36 Pixel für Pixel die Wellenlänge und speichern sie im $wave$ -Element i ab. Nachdem das 5781 mal gemacht wurde, ist unser numpy-Array $wave$ mit den Wellenlängen gefüllt (mit Zahlen im float-Format, also Gleitkommazahlen).

In Zeile 41 schließen wir das fits-File, wir benötigen es nicht mehr.

Dann beginnen wir in Zeile 44 den Spektrumplot zu programmieren. Dies geschieht nach den Regeln, wie sie in der Dokumentation von matplotlib beschrieben sind²⁵.

Betrachten wir den Plot des Spektrums, das unser Programm nun erzeugt hat (Abb. 6.3). Beachten Sie, dass wir die Einstellungen für die IPython-Konsole (siehe Abb. 4.2) so gewählt haben, dass die Grafik in einem separaten Fenster ausgegeben wird. Das hat den Vorteil, dass dieses Fenster aktiv ist. Wir können die Grafik im Fenster manipulieren (zoomen, Achsen verschieben, Beschriftungen und

²⁴Die fits-Dateien enthalten neben dem Header die eigentlichen Daten, hier in $sp[0].data$. Das ist einfach eine Zeile mit $NAXIS1$ Werten, nämlich den Spektrumintensitäten (Flux). Mit den obigen 4 Headerwerten lassen sich dann die zugehörigen Wellenlängen berechnen. Beispielsweise für das 100te Pixel durch

$$\text{Wellenlänge}[99] = CRVAL1 + (99 - CRPIX1 + 1) * CDELTA1$$

Die Indexzählung beginnt in Python mit 0, also ist $\text{Wellenlänge}[0] = CRVAL1 + (0 - CRPIX1 + 1) * CDELTA1 =$ die Startwellenlänge des Spektrums auf der kurzwelligen Seite.

²⁵https://matplotlib.org/api/pyplot_summary.html, <https://matplotlib.org/resources/index.html>

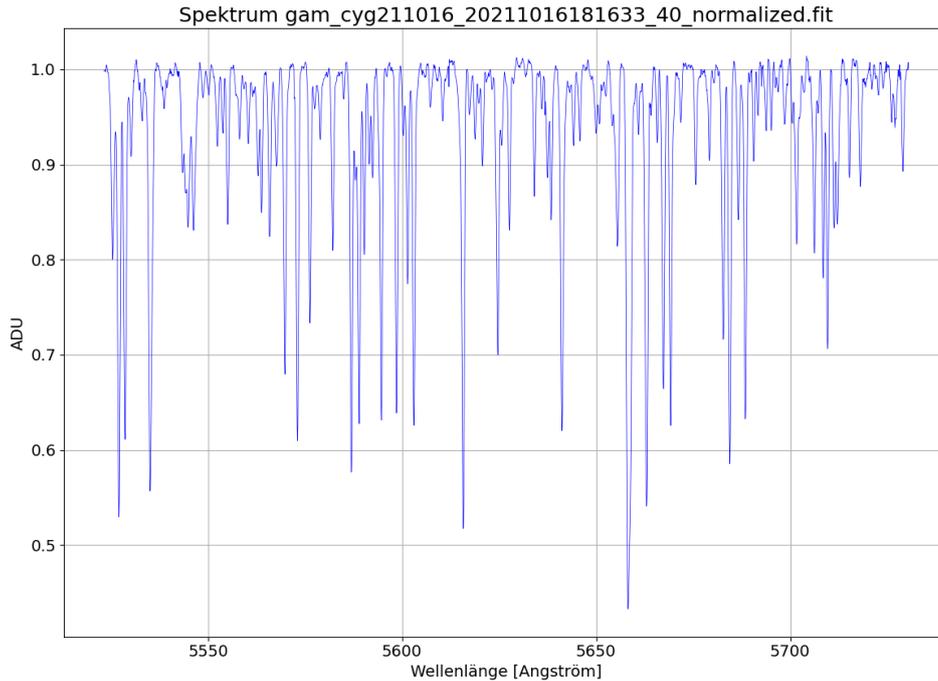


Figure 6.3: Durch das Skript 'spektrum_ansehen.py' erzeugter Plot eines Spektrums.

Kurvenstile ändern) und die gewünschten Ergebnisse manuell z.B. als PNG abspeichern. Das ginge nicht, wenn wir die Grafik „inline“ in die Konsole integrieren würden.

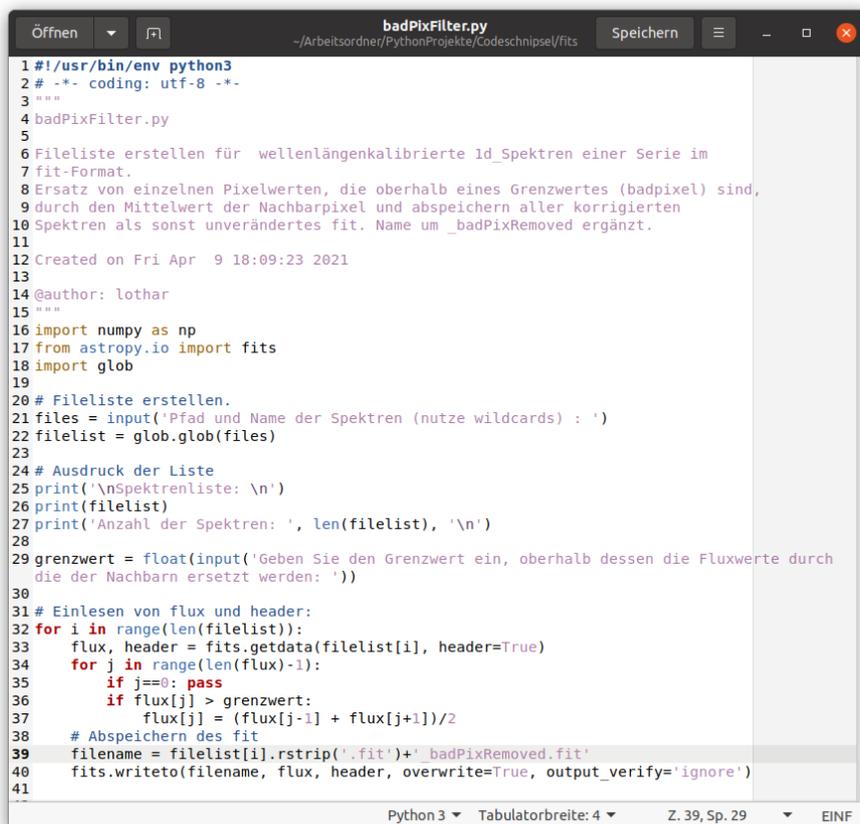
Part III.

Manipulation von 1d-fits-Dateien (1d-Spektren)

Alle für Spektrenserien konzipierten Skripte, die im folgenden vorgestellt werden, können auch für ein einzelnes Spektrum verwendet werden, wenn man als Zeichenfolge für den Spektrenname nicht den Namensstamm (Zeichenfolge mit wildcards) eingibt, sondern lediglich die komplette Bezeichnung für das einzelne Spektrum.

7. Badpixel-Filter

Oft hat man in reduzierten 1d-Rohspektren einzelne Pixel mit extrem hohen ADU (Fluxwerte), welche die benachbarten Pixel um ein Zehnfaches übersteigen und offensichtlich von heißen Pixeln herrühren. Diese einpixeligen Artefakte lassen sich durch das Skript *badPixFilter.py* für eine ganze Serie von Spektren im fits-Format beseitigen. Die von heißen Pixeln (definiert durch einen Flux-Grenzwert, der im Programm abgefragt wird, Zeile 29 in Abb. 7.1) befreiten Spektren werden mit dem Namenszusatz „_badPixRemoved“ als fit-Datei gespeichert.



```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4badPixFilter.py
5
6Fileliste erstellen für wellenlängenkalibrierte 1d-Spektren einer Serie im
7fit-Format.
8Ersatz von einzelnen Pixelwerten, die oberhalb eines Grenzwertes (badpixel) sind,
9durch den Mittelwert der Nachbarpixel und abspeichern aller korrigierten
10Spektren als sonst unverändertes fit. Name um _badPixRemoved ergänzt.
11
12Created on Fri Apr 9 18:09:23 2021
13
14@author: lothar
15"""
16import numpy as np
17from astropy.io import fits
18import glob
19
20# Fileliste erstellen.
21files = input('Pfad und Name der Spektren (nutze wildcards) : ')
22filelist = glob.glob(files)
23
24# Ausdruck der Liste
25print('\nSpektrenliste: \n')
26print(filelist)
27print('Anzahl der Spektren: ', len(filelist), '\n')
28
29grenzwert = float(input('Geben Sie den Grenzwert ein, oberhalb dessen die Fluxwerte durch
die der Nachbarn ersetzt werden: '))
30
31# Einlesen von flux und header:
32for i in range(len(filelist)):
33    flux, header = fits.getdata(filelist[i], header=True)
34    for j in range(len(flux)-1):
35        if j==0: pass
36        if flux[j] > grenzwert:
37            flux[j] = (flux[j-1] + flux[j+1])/2
38    # Abspeichern des fit
39    filename = filelist[i].rstrip('.fit')+'_badPixRemoved.fit'
40    fits.writeto(filename, flux, header, overwrite=True, output_verify='ignore')
41
```

Figure 7.1: Skript zur Beseitigung einpixeliger Artefakte (heisse Pixel) in einem 1d-Spektrum im fits-Format.

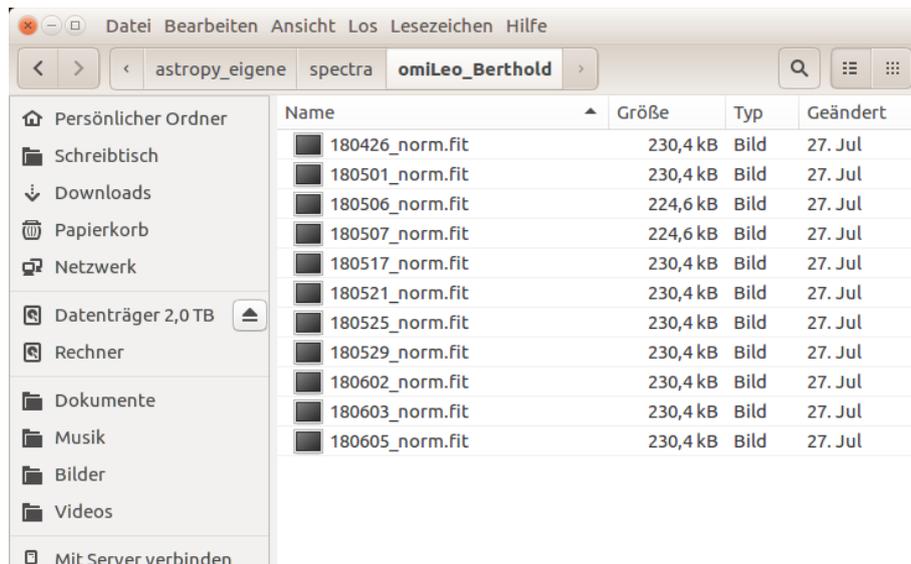


Figure 8.1: Zeitserie von normierten 1d-Echellespektren von omi Leo.

8. Kontrolle und Korrektur des Headers einer Serie von 1d-Spektren im fits-Format

Häufig nimmt man Spektren eines Objekts über längere Zeiträume auf und sammelt sie in reduzierter Form als 1d-Spektren im fits-Format. Dabei können durch Wechsel der Datenreduktionssoftware, der Datenaquisitionsssoftware, eigener Entscheidungen etc. uneinheitliche Header entstehen: Headerkeywords können unterschiedlich sein oder fehlen und vieles mehr.

Dann wünscht man sich eine Software, mit dem man alle Spektren des Objekts auf einen Rutsch bzgl. ihres Headers vereinheitlichen kann.

Nützlich ist, wenn man alle Spektren in einem Ordner als Kopie der Originaldateien vereint hat und ihnen Namen gibt, die einheitlich und informativ aufgebaut sind, z.B. der Filenamen mit dem Jahr/Monat/Tag beginnt. Ein Beispiel ist in Abb. 8.1 gezeigt. Das erleichtert die Eingabe ihres Namens mit Verwendung von wildcards und die zeitliche Ordnung, indem einfach nach Namen alphabetisch sortiert wird.

Nachfolgend besprechen wir ein Python3-Programm namens *HeaderanzeigeUndKorrektur.py*, das solche Serien einliest, ihren Header überprüft und ergänzt oder ändert und die im Header geänderten fits-files wieder abspeichert (und dabei die eingelesenen Dateien überschreibt).

Das Skript in Abb. 8.2 beginnt wie immer mit der Festlegung der Python-Umgebung in Form einer magischen Zeile (Zeile 1) und die Festlegung des Zeichen-Codes (Zeile 2). Dann folgt die Beschreibung in dreifachem Hochkomma oder Hochstrich. Importiert wird das Modul *glob*²⁶ und wieder *fits* aus *astropy.io*. Grafische Hilfsmittel (*matplotlib*) sind diesmal nicht erforderlich.

Aus dem Pfad zu den Dateien (unter Verwendung von wildcards ausgewählt) wird eine Liste der Dateien erstellt, die in Zeile 23 alphabetisch sortiert wird. Zur Kontrolle werden die Dateinamen in der Konsole ausgegeben und ihre Anzahl.

Ab Zeile 29 wird die erste Datei geöffnet und Informationen und ihr gesamter Header in der Konsole ausgegeben. Hier kann man nun prüfen, ob die Headerdaten für die weiters geplanten Auswertungen der Zeitserie vollständig und richtig sind.

Falls der Header nicht den Wünschen entspricht, kann er geändert oder um weitere Headereinträge ergänzt werden. Das geschieht ab Zeile 36. Der User wird gefragt, ob er Änderungen durchführen möchte, wobei nach den Namen der Variablen im Header gefragt wird und nach deren Werten, wobei bei den Werten zwischen Zahlen und Wörtern (strings) unterschieden wird. Wenn ja werden die Headerdaten geändert oder ergänzt und anschließend nach weiteren Änderungen gefragt. Hat man

²⁶ Um zu wissen, welche Module in Python es überhaupt gibt und was sie können, ist es wichtig, immer wieder die Python-Dokumentation zu durchforsten. Insbesondere den Python Package Index (*pypi*), in <https://pypi.org/> zu finden.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 HeaderanzeigeUndKorrektur_Serie.py
5
6 Erzeugung einer Spektralliste der 1d-Spektren im fits-Format in einem Ordner,
7 Korrektur/Ergänzung des Headers.
8
9 Created on Fr Jul 19 14:48:03 2018
10 @author: lothar
11 """
12
13 import glob
14 from astropy.io import fits
15
16 # filelist erstellen
17 # Pfad und Name der Spektren eingeben
18 files = input("Geben Sie Pfad und Namen zu den Spektren ein: ")
19 filelist = glob.glob(files)
20
21 # Sortierung
22 filelist.sort()
23
24 # Ausdruck der Liste
25 print("\nSpektralliste: \n", filelist, "\n")
26 print("Anzahl der Spektren: ", len(filelist), "\n")
27
28 # Header-Check erstes Spektrum
29 hdul = fits.open(filelist[0])
30 print("\nInfo zum ersten Spektrum der Liste:")
31 hdul.info()
32 print("\nKompletter Header des ersten Spektrums der Liste:")
33 print("\n", repr(fits.getheader(filelist[0], 0)))
34
35
36 Eingabe = input(
37     "W möchten Sie für alle Spektren der Serie Headerdaten ändern oder ergänzen? \
38     Wenn nein 0 eingeben, wenn Zahl 1, wenn String 2: ")
39
40 k = 0
41 KW = []
42 Wert = []
43 while int(Eingabe) >= 1:
44     KW.append(input("Eingabe des Header-Keywords: "))
45     if Eingabe == "1":
46         Wert.append(float(input("Eingabe des Wertes dazu: ")))
47     elif Eingabe == "2":
48         Wert.append(input("Eingabe des Wertes dazu: "))
49     else:
50         print("Falsche Eingabe")
51         break
52     Eingabe = input(
53         "Weitere Headereinträge ändern? Wenn nein Eingabe von 0, Wenn ja, Eingabe von
54     1 oder 2: ")
55     k += 1
56
57
58 for i in range(len(filelist)):
59     g = fits.open(filelist[i])
60     for j in range(len(KW)):
61         g[0].header[KW[j]] = Wert[j]
62     fits.update(filelist[i], g[0].data, g[0].header)
63     g.close()

```

Figure 8.2: Programmlisting Header eines 1d-Spektrums (im fits-Format) anzeigen, ändern, aktualisieren.

alle eingegeben, antwortet man mit „0“ (Zeile 53). Anschließend werden alle fits-Dateien der Serie upgedatet (Schleife ab Zeile 58).

9. Erzeugen eines definierten Wellenlängenausschnitts aus einem fits-1d-Spektrum

Manchmal möchte man aus einem „langen“ 1d-Spektrum (z.B. Echellespektrum) nur einen definierten Wellenlängenbereich weiter verarbeiten, beispielsweise für Kreuzkorrelationen oder Vergleiche in einer Zeitserie. Dann ist ein Programm erwünscht, das einen solchen Ausschnitt („crop“) erzeugen und das verkürzte Spektrum als .fit und/oder als .dat mit ergänztem Namen abspeichern kann²⁷.

Der Python3-Code „*Crop_Ausschnitt_fits.py*“ leistet diese Aufgaben. Es erzeugt aus einem Spektrum, beispielsweise Deneb.fit, zwei neue Dateien Deneb_6000_6700.fit und Deneb_6000_6700.dat und zeigt das auf den Wellenlängenbereich 6000 bis 6700 Angström verkürzte Spektrum grafisch an. Der Wellenlängenbereich ist natürlich frei wählbar. In der fit-Datei sind die Headereinträge CVAL1 (Start der Wellenlänge) und NAXIS1 (Anzahl der Pixel) entsprechend angepasst. In der dat-Datei gibt es zwei tab-getrennte Spalten, überschrieben mit 'WAVE' und 'FLUX'.

Das Programm plottet auch das Originalspektrum und das verkürzte in 2 Grafiken, die auf Wunsch auch als PNG gespeichert werden können (Auskommentieren der Zeilen 75 und/oder 104, durch Setzen des Zeichens # an den Anfang der Zeilen).

Das Programm ist in Abb. 9.1 vorgestellt.

10. Zeitreihen von 1d-Spektren im fits-Format auf einen Wellenlängenbereich beschneiden und grafisch darstellen

Nach längerer Beobachtung von Objekten hat man den Wunsch, die 1d-Spektren visuell miteinander zu vergleichen. Dazu gehört ein Plot der Spektren übereinander und versetzt übereinander. Das nachfolgend vorgestellte Skript erledigt diese Aufgabe. Das Programmlisting ist in Abb. 10.1 zu sehen. Die fits-Dateien werden aus einem Ordner eingelesen, ein Wellenlängenbereich gewählt und dann werden die beschnittenen 1d-Dateien als neue fits abgespeichert (falls das nicht gewünscht wird, einfach Zeile 56 auskommentieren (# davor setzen)). In einem Plot werden sie übereinander und in in

²⁷In Abschnitt 13 wird ein Skript vorgestellt, das eine ganze Serie von 1d-fits-Spektren auf einen Wellenlängenbereich beschneidet und gleichzeitig alle Spektren auf eine gemeinsame Schrittweite rebinnt und die erzeugten Spektren als ascii-Datei und als fits-Datei abspeichert.

```

Crop_Ausschnitt_fits.py - /home/lothar/Arbeitsordner/PythonProjekte/Co...
File Edit Format Run Options Window Help
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Crop_Ausschnitt_fits.py
5
6 Ansehen eines wellenlängenkalibrierten 1d-Spektrums,
7 Bereich in der Wellenlängenskala wählen, der im erzeugten Spektrum enthalten
8 sein soll.
9 Anzeige dieses Spektrumsausschnitts als png.
10 Schreiben des neuen fit mit Angabe des Wellenlängenbereichs im Namen und
11 speichern einer Grafik des neuen Spektrums als pdf, falls erwünscht.
12 Abspeichern als .fit und als .dat (Ascii-Datei mit "Tab" als Trennzeichen).
13
14 Stand 20180822
15 @author: lothar
16 """
17
18 import numpy as np
19 from astropy.io import fits
20 from astropy.io import ascii
21 import matplotlib.pyplot as plt
22
23 # Pfad und Name der Spektren, files bitte anpassen
24 file = input('Pfad und Datei-Bezeichnung eingeben: ')
25
26 # Einlesen von Header und Daten
27 flux, header = fits.getdata(file, header=True)
28
29 print('Minimum und Maximum im flux: ', flux.min(), ' ', flux.max())
30
31 # Header-Check Spektrum
32 header_flag = input('Möchten Sie den header komplett sehen? j/n: ')
33 if header_flag == 'j':
34     print('Headerdaten: ')
35     print(header)
36
37 # Prüfung auf die nötigen header-Einträge
38 print('\nAusgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
39 if 'NAXIS' in header:
40     print('Dimension, NAXIS: ', header['NAXIS'])
41 else:
42     print('Das ist kein 1d-Spektrum !')
43 if 'NAXIS1' in header:
44     nax = header['NAXIS1']
45     print('Anzahl der Werte (Abszisse), NAXIS1: ', nax)
46 else:
47     print('NAXIS1 fehlt im header !')
48 if 'CRVAL1' in header:
49     crval = header['CRVAL1']
50     print('Anfangs-Wellenlänge, CRVAL1: ', crval)
51 else:
52     print('CRVAL1 fehlt im header !')
53 if 'CDELTA1' in header:
54     cdel = header['CDELTA1']
55     print('Schrittweite der Wellenlänge, CDELTA1: ', cdel)
56 else:
57
58 else:
59     print('CDELTA1 fehlt im header !')
60
61 # Erzeugen eines numpy-Arrays mit den Wellenlängen des Spektrums
62 wave = np.ones(max, dtype=float)
63 crval = crval + (1 - header['CRPIX1']) * cdel
64 for i in range(max):
65     wave[i] = crval + i*cdel
66
67 # In der Liste wave sind die Wellenlängen der Pixel enthalten
68 # In der Liste flux die zugehörigen Intensitäten
69
70 # Plot gesamtes Spektrum
71 plt.style.use('seaborn-white')
72 fig = plt.figure(1, figsize=(14, 10))
73 plt.plot(wave, flux)
74 plt.xlabel('Wellenlänge [Angström]', fontsize=15)
75 plt.xticks(fontsize=15)
76 plt.ylabel('ADU', fontsize=15)
77 plt.yticks(fontsize=15)
78 plt.title('Spektrum '+file, fontsize=20)
79 plt.grid(True)
80
81 # fig.savefig(file.strip('.fit')+'.pdf')
82 # Kann aktiviert werden, wenn pdf der Grafik erwünscht.
83
84 # Erzeugen des Spektrumsausschnitts, plot und abspeichern als fit
85 print('\nAngabe des zu übernehmenden Wellenlängenbereichs ')
86 a = float(input('Anfang des Spektrums: '))
87 b = float(input('Ende des Spektrums: '))
88 aindex = int((a - crval) / cdel)
89 bindex = int((b - crval) / cdel)
90 filename = file.strip('.fit')+'_'+str(a)+'_'+str(b)+'.fit'
91 newflux = flux[aindex:bindex]
92 header['CRVAL1'] = crval + aindex * cdel
93 header['NAXIS1'] = bindex - aindex
94 header['CRPIX1'] = 1.
95 fits.writeto(filename, newflux, header, overwrite=True, output_verify='ignore')
96
97 # Abspeichern als ascii-File (.dat)
98 newwave = wave[aindex:bindex]
99 filename_dat = file.strip('.fit')+'_'+str(a)+'_'+str(b)+'.dat'
100 ascii.write([newwave, newflux], filename_dat, overwrite=True,
101            names=['WAVE', 'NFLUX'], format='tab')
102
103 # Plot
104 fig = plt.figure(2, figsize=(14, 10))
105 plt.plot(newwave, newflux)
106 plt.xlabel('Wellenlänge [Angström]', fontsize=18)
107 plt.xticks(fontsize=16)
108 plt.ylabel('ADU', fontsize=18)
109 plt.yticks(fontsize=16)
110 plt.title('Spektrum del Cep', fontsize=20)
111 plt.grid(True)
112 fig.savefig(filename.strip('.fit')+'.pdf')
113 # Kann durch Auskommentieren aktiviert werden, wenn pdf der Grafik erwünscht.

```

Figure 9.1: Programmlisting von „*Crop_Ausschnitt_fits.py*“.

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4timeseries.py
5
6Create a file list for wavelength calibrated 1d_spectra of a time
7series.
8Cut out a defined wavelength range from the spectra.
9Save as new fit file.
10Plot the selection into two plots (overplot and offset plots).
11
12release 24.7.2018
13@author: Lothar Schanne
14"""
15import numpy as np
16from astropy.io import fits
17import matplotlib.pyplot as plt
18import glob
19
20# Create file list. All spectra in one (sub)folder.
21files = input('Path and name of the file (use wildcards) : ')
22filelist = glob.glob(files)
23
24# Alphabetical sorting. If the name is correct, this results in a
25# temporal order
26filelist.sort()
27
28# Print the list
29print('\nList of spectra: \n')
30print('Number of spectra: ', len(filelist), '\n')
31
32# Selection of the wavelength range to be extracted:
33print('wavelength range to be extracted: ')
34a = float(input('Begin: '))
35b = float(input('End: '))
36
37fig = plt.figure(1, figsize=(7,10))
38ax1, ax2 = fig.subplots(2,1, sharex=True, sharey=False)
39
40# Read header and flux
41for i in range(len(filelist)):
42    print(filelist[i], ':')
43    flux, header = fits.getdata(filelist[i], header=True)
44    # Generate the spectrum sections and save them as fit:
45    step = header['CDELTA1']
46    refpix = header['CRPIX1']
47    wave_erstesPix = header['CRVAL1'] - step*(refpix - 1)
48    aindex = int((a - wave_erstesPix) / step)
49    bindex = int((b - wave_erstesPix) / step)
50    begin = wave_erstesPix + aindex * step
51    filename = filelist[i].strip('.fit')+'_'+str(int(a))-
52    '+'+str(int(b))+'.fit'
53    newflux = flux[aindex:bindex]
54    header['CRVAL1'] = begin
55    header['NAXIS1'] = bindex - aindex
56    header['CRPIX1'] = 1
57    fits.writeto(filename, newflux, header, overwrite=True,
58    output_verify='ignore')
59
60# Graphic with all spectra cutouts:
61newwave = np.ones(bindex - aindex, dtype=float)
62for k in range(bindex - aindex):
63    newwave[k] = begin + k * step
64ax1.plot(newwave, newflux)
65ax2.plot(newwave, newflux+i*1.0, '-', label=filename)
66
67# Customize plot properties (grid, labels, etc.):
68fig.suptitle('Time Series of '+header['OBJECT'])
69ax1.grid(True)
70ax2.grid(True)
71ax2.set_xlabel('Wavelength in Angström')
72# #Legend can be adjusted, e.g. font size (fontsize)
73plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
74# ncol=4, mode="expand", borderaxespad=1., fontsize=6)
75fig.savefig(header['OBJECT']+'_timeseries.pdf')
76fig.savefig(header['OBJECT']+'_timeseries.png')

```

Figure 10.1: Skript zum Beschneiden und Plotten von Zeitreihen von 1d-Spektren (.fits).

einem zweiten übereinander, aber senkrecht um einen wählbaren Betrag versetzt geplottet. Die Grafik wird auch als PDF und PNG abgespeichert.

In Abb. 10.2 ist ein Plot aus 11 Spektren gezeigt.

11. Übersetzen eines 1d-Spektrums im fits-Format in ascii-Dateien

Gelegentlich möchte man 1d-Spektren in Programmen verarbeiten, die fits-Formate nicht lesen können (z.B. Excel). Dann ist es vorteilhaft, die Spektren als Ascii-Datei vorliegen zu haben, z.B. tab-separiert (.dat) oder Komma-separiert (.csv).

Das leistet das Python3-Skript „*fit_in_csv_und_dat.py*“. In Abb. 11.1 ist das Programmlisting gezeigt. In der erzeugten dat- und csv-Datei gibt es zwei Spalten von Fließkommazahlen (type float mit Punkt als Dezimalzeichen), überschrieben mit 'WAVE' und 'FLUX'.

12. Erzeugen eines gemeinsamen Wellenlängenausschnitts einer Serie von 1d-Spektren im fits-Format und abspeichern als fits- und als ascii-Dateien

Häufig benötigt man Spektren im ascii-Format als Wertetabellen, und zwar gleich für eine ganze Serie von 1d-Spektren im fits-Format. Das erledigt das Skript *Crop_Auschnitt_Zeitserie_fits_dat.py*, das in Abb. 12.1 gelistet ist.

Das Skript gibt nach Eingabe des Pfades und Dateinamenstamms (wildcards benutzen !) zuerst einen Plot aller Spektren aus (ab Zeile 34), der auf Wunsch auch gespeichert werden kann (Zeilen 56 und 57 auskommentieren). Danach wird der gemeinsame Wellenlängenbereich aller Spektren ermittelt und in der Konsole ausgedruckt. Anschließend wird der gewünschte Wellenlängenausschnitt abgefragt, auf den die Spektren beschnitten werden sollen (ab Zeile 73). Diese Ausschnitte werden dann berechnet (Schleife ab Zeile 77) und die berechneten beschnittenen Spektren als fits-Datei sowie als csv- und als tab-Datei abgespeichert.

Möchte man auch die Header der fits-Dateien getrennt als ascii-Dateien zur Verfügung haben, kann das Skript *Headerprint_Serie_csv.py* verwenden (Abb. 12.2), welches alle Header der Serie einliest

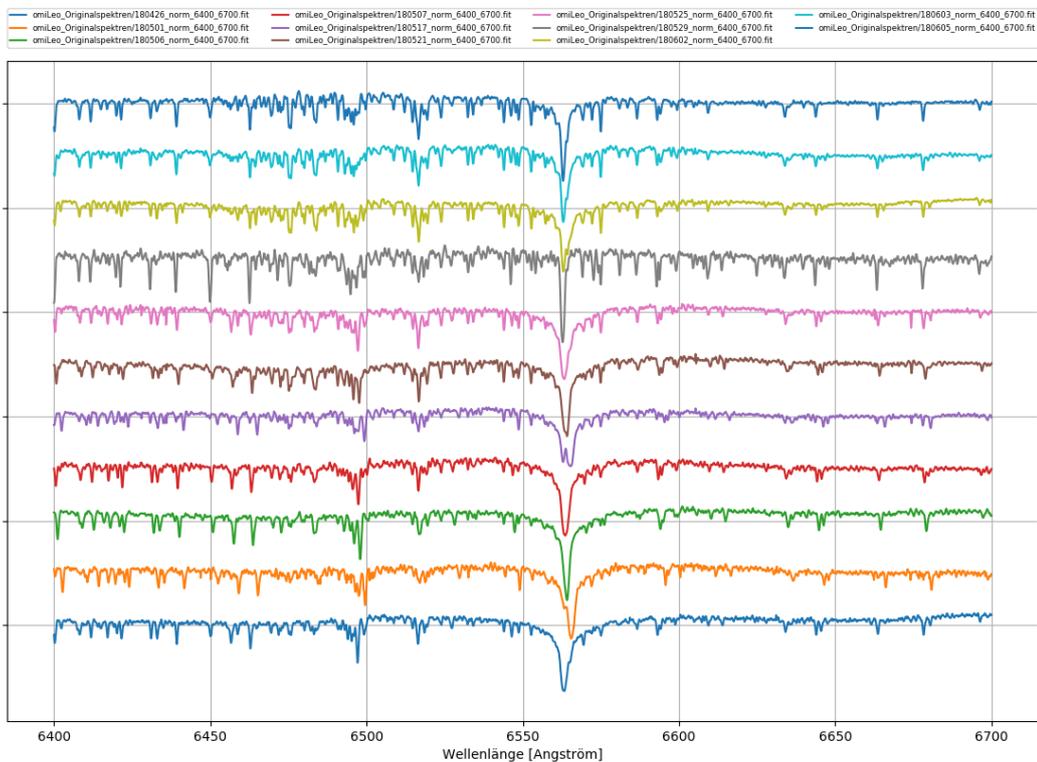
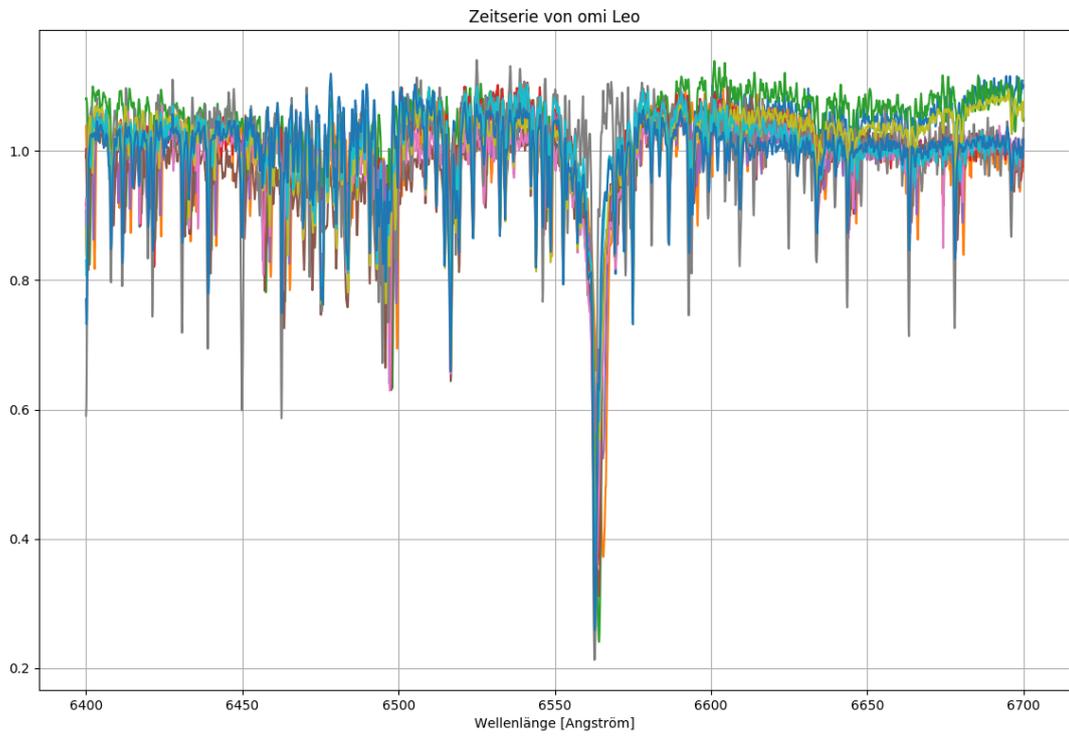


Figure 10.2: Plot einer Zeitreihe von 1d-Spektren. Oben alle übereinander geplottet, unten vertikal versetzt.

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4fit_in_csv_and_dat.py
5
6Conversion of a wavelength calibrated 1d spectrum in fits format into
7Text format .csv (comma-separated) and .dat format (tab-separated).
8With column headings 'WAVE' and 'FLUX'.
9
10Release 20180815
11@author: Lothar Schanne
12"""
13
14import numpy as np
15from astropy.io import fits
16from astropy.io import ascii
17
18# Pfad und Name des 1d-fit-Spektrums
19file = input('Path and name of the fits-file: ')
20
21# Einlesen von Header und Daten
22flux, header = fits.getdata(file, header=True)
23
24print('Minimum and Maximum in flux: ', flux.min(), ' ', flux.max())
25
26# Header-Check Spectrum
27header_flag = input('Would you like to see the header completely? y/n:')
28if header_flag == 'y':
29    print('Header: ')
30    print(header)
31
32# Check for the necessary header entries
33print('\nOutput of the header entries required for wavelength calculation:')
34if 'NAXIS' in header:
35    print('Dimension, NAXIS:          ', header['NAXIS'])
36else:
37    print('That is no 1d-Spectrum !')
38if 'NAXIS1' in header:
39    nax = header['NAXIS1']
40    print('Number of values (abscissa), NAXIS1:    ', nax)
41else:
42    print('NAXIS1 is missing in header !')
43if 'CRVAL1' in header:
44    crval = header['CRVAL1']
45    print('Begin of wavelength-scale, CRVAL1:      ', crval)
46else:
47    print('CRVAL1 is missing in header !')
48if 'CDELTA1' in header:
49    cdel = header['CDELTA1']
50    print('Wavelength increment, CDELTA1:    ', cdel)
51else:
52    print('CDELTA1 is missing in header !')
53
54# Generation of a numpy array with the wavelengths of the spectrum
55wave = np.ones(nax, dtype=float)
56for i in range(nax):
57    wave[i] = crval + (i - header['CRPIX1'] + 1) * cdel
58
59# Writing the csv- and dat-files
60ascii.write([wave, flux], file.strip('.fit')+'.csv', overwrite=True,
61            names=['WAVE', 'FLUX'], format='csv')
62ascii.write([wave, flux], file.strip('.fit')+'.dat', overwrite=True,
63            names=['WAVE', 'FLUX'], format='tab')
```

Figure 11.1: Skript fit_in_csv_and_dat.py zur Umwandlung von 1d-Spektren im fits-Format in Ascii-Dateien.

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3 """
4 Fileliste erstellen für wellenlängenkalibrierte 1d Spektren im fits-Format
5 einer Zeitreihe. Plotten der Spektren. Beschneiden aller Spektren auf einen
6 wählbaren Wellenlängenbereich und abspeichern aller beschnittenen als fit oder
7 als ascii-Datei (tab und csv) mit Wellenlängenbereich im Dateinamen.
8
9
10 @author: lothar
11 20.1.2022
12 """
13
14 import numpy as np
15 from astropy.io import fits
16 import matplotlib.pyplot as plt
17 import glob
18
19
20 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
21 files = input('Pfad und Name der Spektren (nutze wildcards) : ')
22 filelist = glob.glob(files)
23
24 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
25 # zeitliche Ordnung
26 filelist.sort()
27
28 # Ausdruck der Liste
29 print('\nSpektrenliste: \n')
30 print(filelist)
31 print('Anzahl der Spektren: ', len(filelist), '\n')
32
33
34 fig = plt.figure(figsize=(14, 20))
35 # Einlesen von flux und header:
36 for i in range(len(filelist)):
37     flux, header = fits.getdata(filelist[i], header=True)
38     # Grafik mit allen Spektrenausschnitten:
39     wave = np.zeros(header['NAXIS1'], dtype=float)
40     for k in range(len(wave)):
41         wave[k] = header['CRVAL1'] + (k-header['CRPIX1']+1)*header['CDELT1']
42     plt.plot(wave, flux, linewidth=1)
43
44
45 # Ploteigenschaften (Gitter, Beschriftungen etc.) anpassen:
46 plt.title('Zeitserie von '+header['OBJECT'])
47 plt.grid(True)
48 plt.xlabel('Wellenlänge [Angström]')
49 # Legende kann angepasst werden, z.B. Schriftgröße (fontsize)
50 plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
51           ncol=4, mode='expand', borderaxespad=0., fontsize=6)
52
53 plt.show(block=False)

```

```

54
55 # Grafiken speichern
56 #fig.savefig(header['OBJECT']+'_Zeitserie.pdf')
57 #fig.savefig(header['OBJECT']+'_Zeitserie.png')
58
59
60 # Generate the spectrum section, plot and save as fit
61 min = 0
62 max = 10000
63
64 for i in range(len(filelist)):
65     sp = fits.open(filelist[i])
66     if sp[0].header['CRVAL1'] > min:
67         min = sp[0].header['CRVAL1']
68     if sp[0].header['CRVAL1'] + sp[0].header['CDELT1']*sp[0].header['NAXIS1'] <
69         max:
70         max = sp[0].header['CRVAL1']
71         +sp[0].header['CDELT1']*sp[0].header['NAXIS1']
72     sp.close()
73 print('\nGemeinsamer Wellenlängenbereich: ', min, max)
74
75 # Specification of the wavelength range to be transferred *)
76 a = float(input('Begin: '))
77 b = float(input('End: '))
78
79 for i in range(len(filelist)):
80     sp = fits.open(filelist[i])
81     step = sp[0].header['CDELT1']
82     refpix = sp[0].header['CRPIX1']
83     wave_erstesPix = sp[0].header['CRVAL1'] - step*(refpix - 1)
84     aindex = int((a - wave_erstesPix) / step)
85     bindex = int((b - wave_erstesPix) / step)
86     a = sp[0].header['CRVAL1'] + aindex * step
87     b = sp[0].header['CRVAL1'] + bindex * step
88     filename = filelist[i].rsplit('.')[-1]+'_'+str(int(a))+'_'+str(int(b))+'_fit'
89     newflux = sp[0].data[aindex:bindex]
90     sp[0].header['CRVAL1'] = a
91     sp[0].header['NAXIS1'] = bindex - aindex
92     sp[0].header['CRPIX1'] = 1
93     fits.writeto(filename, newflux, sp[0].header, overwrite=True,
94                 output_verify='ignore')
95     newwave = np.zeros(sp[0].header['NAXIS1'], dtype=float)
96     for k in range(len(newwave)):
97         newwave[k] = a + k * step
98     # filename = filelist[i].rsplit('.')[-1]+'_'+str(int(a))+'_'+str(int(b))-
99     # '+'.csv'
100     # ascii.write([newwave, newflux], filename, overwrite=True,
101                 names=['WAVE', 'FLUX'], format='csv')
102     filename = filelist[i].rsplit('.')[-1]+'_'+str(int(a))+'_'+str(int(b))+'_dat'
103     ascii.write([newwave, newflux], filename, overwrite=True,
104                 names=['WAVE', 'FLUX'], format='tab')
105     sp.close()

```

Figure 12.1: Code des Skripts Crop_Auschnitt_Zeitserie_fits_dat.py zur Beschneidung einer Serie von 1d-Spektren im fits-Format und Abspeicherung der neuen Spektren als fits- und als ascii-Dateien.

und als csv-Datei abspeichert.

13. Rebinnen einer Serie von 1d-Spektren im fits-Format auf eine gemeinsame Schrittweite und einen gemeinsamen Wellenlängenbereich sowie Bildung eines gemittelten Spektrums

Häufig ist es für weitergehende Auswertungen einer Serie von 1d-Spektren nötig, dass diese einen gemeinsamen Wellenlängenbereich und die gleiche Schrittweite haben. Oder man möchte eine Serie von 1d-Spektren zu einem gemittelten Spektrum zusammen fassen. Beides erledigt das Skript *newbinning_mittleresSpek* (Abb. 13.1).

Nach der Angabe von Pfad und Namensstamm der Spektrenserie wird der gemeinsame Wellenlängenbereich und der Bereich der Schrittweiten der Serie ermittelt (Zeilen 43 bis 63). Daraufhin kann man die neue Schrittweite eingeben und den Wellenlängenbereich, für den die Spektren berechnet werden sollen (Zeilen 68 bis 72). In der Schleife ab Zeile 78 werden dann alle Spektren der Serie umgerechnet, und ab Zeile 111 als ascii-Datei und ab Zeile 119 als fits-Datei berechnet und im Arbeitsverzeichnis abgespeichert. Ab Zeile 137 wird das mittlere Spektrum berechnet und als ascii- und als fits-Datei abgespeichert.

Alle so gebildeten Dateien haben die gleichen Wellenlängen (Anfangs- und Endwert sowie alle Pixel stimmen überein), so dass mit ihnen arithmetische Operationen durchgeführt werden können (Addition, Subtraktion).

Falls eine Berechnung des gemittelten Spektrums und/oder sein plotten nicht gewünscht ist, bitte die Codezeilen 137 bis 160 durch Einfügen von # am Anfang der Zeilen zu Kommentaren umwandeln. Falls man nur das mittlere Spektrum berechnen will kann das abspeichern der ascii-Dateien oder der fits-Dateien kann durch kommentieren der betreffenden Codezeilen unterbunden werden (Zeilen 110 bis 117 bzw. 123 bis 132).

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3'''
4lookHeader.py
5
6
7Created on Fr Jul 19 14:48:03 2018
8@author: Lothar Schanne
9'''
10
11import glob
12from astropy.io import fits
13
14# filelist erstellen
15# Pfad und Name der Spektren eingeben
16files = input("Geben Sie Pfad und Namen zu den Spektren ein: ")
17filelist = glob.glob(files)
18
19# Sortierung
20filelist.sort()
21
22# Ausdruck der Liste
23print("\nSpektrenliste: \n", filelist, "\n")
24print("Anzahl der Spektren: ", len(filelist), "\n")
25
26
27for f in range(len(filelist)):
28    g = fits.open(filelist[f])
29
30    print("\nInfo about the spectrum: " + filelist[f])
31    g.info()
32    header = g[0].header
33    headerdict = dict(header)
34    f = open(filelist[f].rstrip(".fit") + "_Header.csv", "w")
35    for i in headerdict:
36        f.write(str(i) + "," + str(headerdict[i]) + "\n")
37    f.close()

```

Figure 12.2: Skript zum Abspeichern der Headerdaten einer 1d-Spektrenserie im fits-Format als ascii-Dateien (csv, kommasepariert).

Part IV.

Umgang mit 1d-Spektren im ASCII-Format

Ascii-Dateien (hier Werte-Tabellen im Text-Format) spielen bei vielen Astro-Programmen eine wichtige Rolle als Datenspeicher. Ein 1d-Spektrum in ascii-Format besteht mindestens aus 2 Spalten, eine für die Wellenlänge, die zweite für den Flux. Es können einfache Zahlen-Spalten (im Dezimalzahlenformat mit Punkt als Dezimalzeichen) ohne jede Überschrift sein, sie können aber auch Spaltenüberschriften tragen.

Wie wir unsere fits-1d-Spektren in solche ascii-Tabellen übersetzen können, um sie dann in einem anderen Programm (wie z.B. Excel, aber auch in Python-Skripten) verarbeiten zu können, haben wir bereits in Teil III gesehen.

Die Umwandlung solcher ascii-Dateien in welche mit anderem Trennzeichen zwischen den Spalten kann mit vielen Editoren erledigt werden. Ebenso das Hinzufügen oder Entfernen von Spaltenüberschriften. Beides wird deshalb hier nicht weiter erläutert. In Python gibt es natürlich auch eine Menge von Möglichkeiten, ascii-files zu lesen und zu verändern²⁸.

14. Plotten eines ascii-1d-Spektrums

Liegt das Spektrum als ascii-File vor, einfach 2 Spalten von Wertepaaren (Wellenlänge und Flux) ohne Überschrift, kann es mit dem einfachen Skript *1d_txt_plotten.py* in Abb. 14.1 geplottet werden.

15. Plotten und erzeugen eines Ausschnitts aus einem ascii-1d-Spektrum

Ähnlich wie in Abschnitt 9 mit fits-Dateien können wir auch aus einem 1d-Spektrum im ascii-Format (.dat, .csv) das Spektrum grafisch darstellen und einen beliebigen Ausschnitt erzeugen (vgl. Abb. 15.1). Das Ergebnis des Skripts *PlotAndCrop_1d_ascii.py* sind eine Grafik des kompletten Spektrums und

²⁸<https://docs.python.org/3/library/csv.html#>

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3 """
4 newbinning_mittleresSpektrum.py
5 Liest einen Spektrenkatalog ein (fits), rebinnt die Spektren und erzeugt
6 tab-Spektren (tab-Tabelle mit den Spaltenbenennungen WAVE und FLUX) sowie
7 fits-Dateien, alle mit gleicher wählbarer Schrittweite und gleichem
8 Wellenlängenbereich. Die Art der Interpolation (linear oder per kubischem
9 Spline) ist durch auskommentieren wählbar. Außerdem wird ein gemittelttes
10 Spektrum berechnet und als tab-Tabelle abgespeichert mit den Spaltennamen
11 WAVE und FLUX. Das gemittelte Spektrum wird auch geplottet.
12 @author: lothar schanne
13 24.01.2022
14 """
15
16 import numpy as np
17 from astropy.io import fits, ascii
18 import matplotlib.pyplot as plt
19 import glob
20 from specutils import Spectrum1D
21 from specutils.manipulation import (
22     LinearInterpolatedResampler,
23     SplineInterpolatedResampler,
24 )
25 from astropy import units as u
26 from astropy.visualization import quantity_support
27 quantity_support()
28
29 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
30 files = input("Pfad und Name der Spektren (nutze wildcards) : ")
31 filelist = glob.glob(files)
32
33 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
34 # zeitliche Ordnung
35 filelist.sort()
36
37 # Ausdruck der Liste
38 print("\nSpektrenliste: \n")
39 print(filelist)
40 print("Anzahl der Spektren: ", len(filelist), "\n")
41
42 # Berechnung und Ausgabe des gemeinsamen Wellenlängenbereichs und der Schrittweiten
43 well_min = 0
44 well_max = 10000
45 step_min = 100
46 step_max = 0
47 for i in range(len(filelist)):
48     sp = fits.open(filelist[i])
49     crval = sp[0].header["CRVAL1"]
50     if "CRPIX1" not in sp[0].header:
51         sp[0].header["CRPIX1"] = 1
52     crpix = sp[0].header["CRPIX1"]
53     cdel = sp[0].header["CDEL11"]
54     wave_erstesPixel = crval - cdel * (crpix - 1)

```

```

54     wave_erstesPixel = crval - cdel * (crpix - 1)
55     if wave_erstesPixel > well_min:
56         well_min = wave_erstesPixel
57     if wave_erstesPixel + cdel * sp[0].header["NAXIS1"] < well_max:
58         well_max = wave_erstesPixel + cdel * sp[0].header["NAXIS1"]
59     if cdel < step_min:
60         step_min = cdel
61     if cdel > step_max:
62         step_max = cdel
63     sp.close()
64
65 print("\nGemeinsamer Wellenlängenbereich: ", well_min, well_max)
66 print("Minimale und maximale Schrittweite: ", step_min, step_max)
67
68 # zu wählende Schrittweite und Wellenlängenbereich
69 newstep = float(input("Geben Sie die gewünschte Schrittweite ein: "))
70 print("\nSpezifikation der wavelength range to be transferred :")
71 a = float(input("Wellenlänge Begin: "))
72 b = float(input("Wellenlänge End: "))
73
74 pixzahl = int((b - a) / newstep)
75 DATA = np.zeros((len(filelist), pixzahl))
76
77 # Berechnung der neuen Spektren der Serie
78 for i in range(len(filelist)):
79     sp = fits.open(filelist[i])
80     crval = sp[0].header["CRVAL1"]
81     if "CRPIX1" not in sp[0].header:
82         sp[0].header["CRPIX1"] = 1
83     crpix = sp[0].header["CRPIX1"]
84     cdel = sp[0].header["CDEL11"]
85     wave_erstesPixel = crval - cdel * (crpix - 1)
86     aindex = int((a - wave_erstesPixel) / cdel)
87     bindex = int((b - wave_erstesPixel) / cdel)
88     newflux = sp[0].data[aindex:bindex]
89     newflux = newflux * u.dimensionless_unscaled
90     newwave = np.zeros(bindex - aindex)
91     for k in range(len(newwave)):
92         newwave[k] = wave_erstesPixel + (aindex + k) * cdel
93     newwave = newwave * u.AA
94
95 # new binning:
96 input_spec = Spectrum1D(spectral_axis=newwave, flux=newflux)
97 new_disp_grid = np.arange(a, b, newstep) * u.AA
98
99 # Interpolation
100 # lineare Interpolation:
101 linear = LinearInterpolatedResampler()
102 new_spec = linear(input_spec, new_disp_grid)
103 # Interpolation per Spline:
104 spline = SplineInterpolatedResampler()
105 new_spec = spline(input_spec, new_disp_grid)
106
107 filename = (

```

```

107     filename = (
108         filelist[i].rsplit(".")[-1][0] + "_" + str(int(a)) + "_" + str(int(b)) + ".dat"
109     )
110     # ascii-Datei speichern
111     ascii.write(
112         [new_spec.spectral_axis, new_spec.flux],
113         filename,
114         overwrite=True,
115         names=["WAVE", "FLUX"],
116         format="tab",
117     )
118     # fits-Datei berechnen und speichern
119     sp[0].header["CRVAL1"] = float(new_spec.spectral_axis[0].value)
120     sp[0].header["CRPIX1"] = 1
121     sp[0].header["NAXIS1"] = len(new_spec.spectral_axis)
122     sp[0].header["CDEL11"] = newstep
123     newfile = (
124         filelist[i].rsplit(".")[-1][0] + "_" + str(int(a)) + "_" + str(int(b)) + ".fit"
125     )
126     fits.writeto(
127         newfile,
128         new_spec.flux.value,
129         sp[0].header,
130         overwrite=True,
131         output_verify="ignore",
132     )
133     for j in range(len(new_spec.flux)):
134         DATA[i, j] = new_spec.flux[j].value
135     sp.close()
136
137 # Berechnung und Abspeichern des mittleren Spektrums
138 obj = input("Geben Sie den Objektname ohne blanks ein: ")
139 mean = np.zeros(len(DATA[1]))
140 for m in range(len(DATA[1])):
141     mean[m] = DATA[:, m].mean()
142 file = obj + "_" + str(int(a)) + "_" + str(int(b))
143 filename = file + "_mean.dat"
144 ascii.write(
145     [new_spec.spectral_axis / u.AA, mean[:]],
146     filename,
147     overwrite=True,
148     names=["WAVE", "FLUX"],
149     format="tab",
150 )
151 filename = file + "_mean.fits"
152 fits.writeto(filename, mean[:], sp[0].header, overwrite=True, output_verify="ignore")
153
154 # Plotten des mittleren Spektrums
155 fig = plt.figure(figsize=(14, 14))
156 plt.xlabel("Wavelength [Angstrom]")
157 plt.ylabel("Flux")
158 plt.title("Mittleres Spektrum " + filename)
159 plt.plot(new_spec.spectral_axis, mean[:])
160 fig.savefig(filename.rstrip(".fits") + ".png")

```

Figure 13.1: Skript zum Rebinning einer Serie von 1d-Spektren im fits-Format auf einen gemeinsamen Wellenlängenbereich und gleiche Schrittweite sowie Mitteln der Spektren.

```

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe
Öffnen ▾ [F4] Speichern

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Das Skript liest ein Spektrum ein, das als ASCII-file gespeichert ist. Die
Daten stehen in 2 Spalten ohne Überschrift.
Pfad/Name werden erfragt.
Das Spektrum wird geplottet.

Stand 20180823
Autor = Lothar Schanne
"""

import matplotlib.pyplot as plt
from astropy.io import ascii

spectrum_name = input('Geben Sie Pfad und Namen des Textfiles ein: ')
spectrum = ascii.read(spectrum_name, guess=True)

# Plotten des Spektrums
fig = plt.figure(figsize=(14, 10))
plt.plot(spectrum['col1'], spectrum['col2'])
plt.xlabel('Wellenlänge [Angström]')
plt.ylabel('ADU')
plt.title(spectrum_name)
plt.grid(True)

```

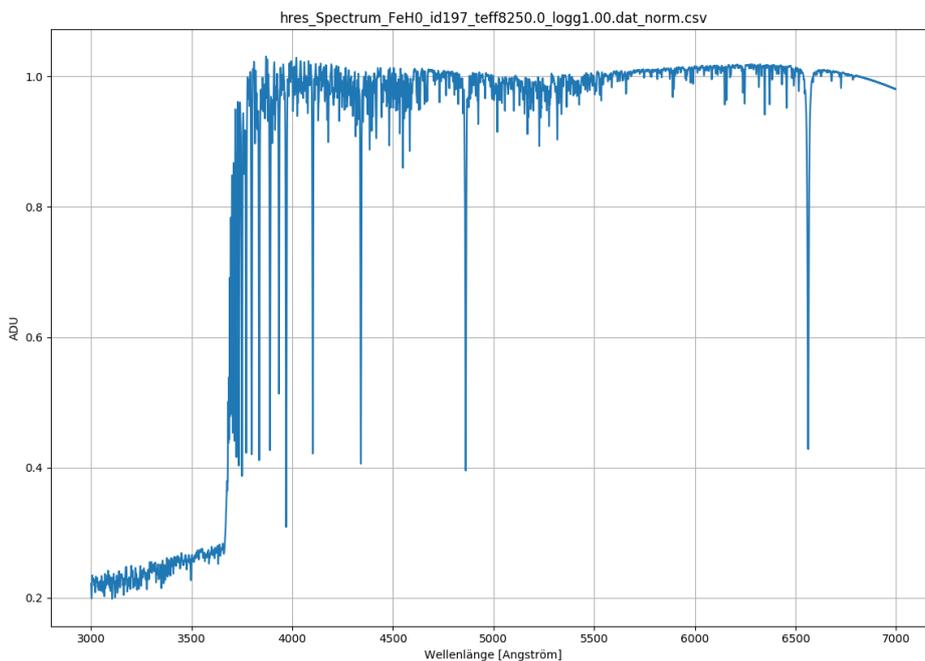


Figure 14.1: Listing von *1d_txt_plotten.py* und Abbildung eines ascii-1d-Spektrums erzeugt durch *1d_txt_plotten.py*.

des erzeugten Ausschnitts (Abb. 15.2) sowie eine .dat-Datei des erzeugten Ausschnitts, entsprechend benannt.

16. Erzeugen von Differenzspektren

Um systematische Unterschiede zwischen Spektren einer Serie und einem Vergleichsspektrum (mittleres Spektrum der Serie oder ein theoretisch gerechnetes Spektrum) zu untersuchen ist es sinnvoll, Differenzspektren zu berechnen, also die Differenz zwischen Spektrum und Vergleichsspektrum.

Um die Subtraktion durchzuführen ist es erforderlich, dass alle verwendeten Spektren den gleichen Wellenlängenbereich und die gleiche Schrittweite besitzen, wie wir sie mit dem Skript in Abschnitt 13 erzeugt haben. Alle Spektren müssen im ascii-Format 'tab' vorliegen.

Das Skript ist in Abb. 16.1 abgebildet.

Part V.

Theoretische Spektren und Konvolution (Faltung) von Modell-1d-Spektren mit Gaussfunktionen

Modellspektren (also theoretisch berechnete), wie man sie aus dem Internet beziehen kann²⁹, besitzen Dispersionen, die sich von den gemessenen meist deutlich unterscheiden. Um solche theoretischen Spektren mit den (eigen)gemessenen vergleichen zu können, ist es sinnvoll die höher aufgelösten Spektren (meist die theoretischen) mit dem als Gaussprofil angenommenen Apparatprofil der weniger aufgelösten zu falten (z.B. in Form der FWHM der schmalsten Linien im gemessenen Spektrum, am besten der terrestrischen Linien) .

17. Konvolvieren eines 1d-fits-Spektrums auf eine bestimmte Auflösung R

Manchmal möchte man ein hoch aufgelöstes 1d-Spektrum im fits-Format auf eine geringere Auflösung R bringen. Das kann mit dem Skript `convol_fits.py` geschehen. Nach dem Einlesen des Spektrums und der Eingabe des gewünschten R (z.B. 10000) wird das knvolvierte Spektrum berechnet und abgespeichert. Der ursprüngliche Dateinamen wird zur Unterscheidung vom Original um den Anhang „`_convolved_R`“ ergänzt.

18. Modifizierung eines 1d-Spektrums im ascii-Format (`convol_dat.py`)

Ein einfaches Python3-Skript zum Falten eines 1d-Spektrums mit einer Gaussfunktion, das in Form einer 2-spaltigen ascii-Datei vorliegt³⁰, ist nachfolgend aufgeführt. Eingegeben wird der Pfad/Name des ascii-Files und der Standardabweichung der Gaussfunktion, mit der gefaltet werden soll (in Vielfachen

²⁹<http://svo2.cab.inta-csic.es/theory/newov/>
<http://pollux.graal.univ-montp2.fr/>
<http://marcs.astro.uu.se/>

³⁰Vorzugsweise tab-separiert und mit den Spaltenüberschriften WAVE und FLUX. Falls das nicht der Fall ist, die Dateien entsprechend ändern oder das Pythonskript anpassen.

```

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe
Öffnen [F4] Speichern

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Das Skript liest ein ascii-file ein und plottet ihn. Der ascifile hat 2 Spalten
mit den Spaltenüberschriften 'WAVE' und 'FLUX'. Die Extension des files sei .dat.
Separierungszeichen wird automatisch ermittelt mit guess=True.

Wahlweise kann einen Ausschnitt (Wellenlängenbereich) erzeugt werden.
Dieser wird als .dat abgespeichert und geplottet.

Created on Sun Jul 29 15:43:51 2018

@author: lothar
"""

import numpy as np
import matplotlib.pyplot as plt
from astropy.io import ascii

# Eingabe des ascii-files
# Dieser muss 2 beliebig aber einheitlich separierte Spalten WAVE und FLUX
# haben
spectrum_name = input('Geben Sie Pfad und Namen des ascii-files ein: ')
spectrum = ascii.read(spectrum_name, guess=True)

fig = plt.figure(figsize=(14, 10))
plt.subplot(2, 1, 1)
plt.plot(spectrum['WAVE'], spectrum['FLUX'])
plt.xlabel('Wellenlänge [Angström]')
plt.ylabel('ADU')
plt.title(spectrum_name)
plt.grid(True)

# Erzeugen des Ausschnitts, plot und abspeichern als ascii-file
antwort = input('Möchten sie einen Ausschnitt erzeugen? j/n: ')
if antwort == 'j':
    print('Angabe des zu übernehmenden Wellenlängenbereichs ')
    aindex = int(input('Anfang des Spektrums: '))
    bindex = int(input('Ende des Spektrums: '))
    filename = spectrum_name.rstrip('.dat')+'_'+str(aindex)+'_'+str(bindex)
    # Indices suchen
    file_wave = np.ones(len(spectrum))
    file_flux = np.ones(len(spectrum))
    for m in range(len(spectrum)):
        file_wave[m] = spectrum['WAVE'][m]
    for m in range(len(spectrum)):
        file_flux[m] = spectrum['FLUX'][m]
    for m in range(len(spectrum)):
        if file_wave[m] > aindex:
            aind = m
            break
    for m in range(len(spectrum)):
        if file_wave[m] > bindex:
            bind = m
            break
    file_wave = file_wave[aind:bind]
    file_flux = file_flux[aind:bind]
    ascii.write([file_wave, file_flux], filename+'.dat', overwrite=True,
               names=['WAVE', 'FLUX'], format='tab')
    plt.subplot(2, 1, 2)
    plt.plot(file_wave, file_flux)
    plt.xlabel('Wellenlänge [Angström]')
    plt.ylabel('ADU')
    plt.title(filename)
    plt.grid(True)
    fig.savefig(filename+'.pdf')
    fig.savefig(filename+'.png')

```

Figure 15.1: Skript zum plotten eines ascii-1d-Spektrums und erzeugen eines Ausschnitts

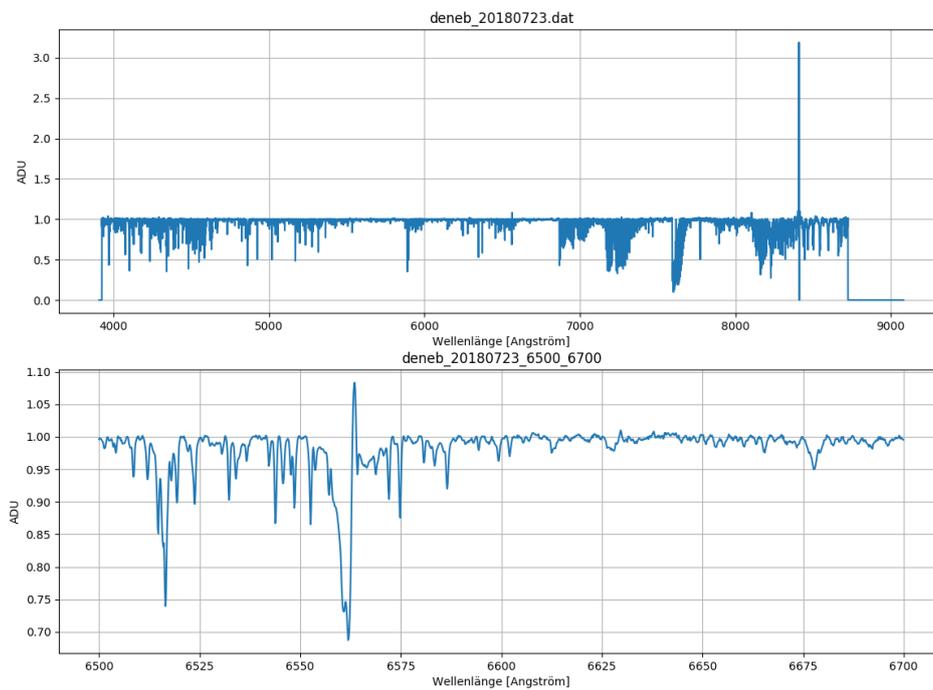


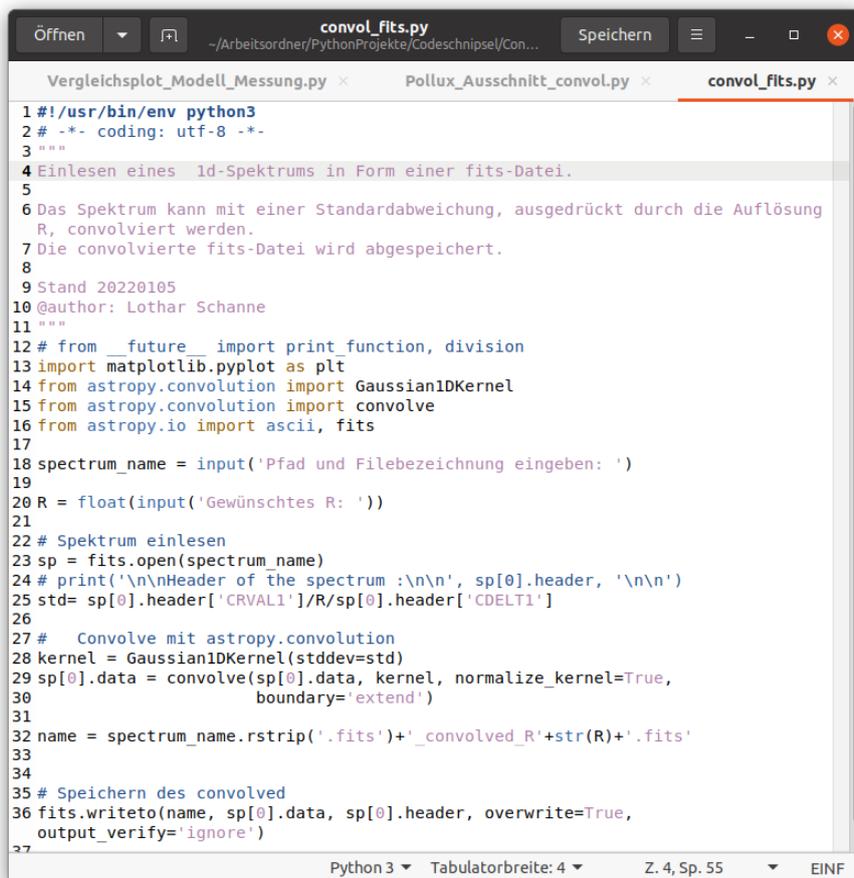
Figure 15.2: Resultierende Grafik des Skripts *PlotAndCrop_1d_ascii.py*. Oben das gesamte Echelle-Spektrum von Deneb, unten ein Ausschnitt von 6500 bis 6700 Angström.

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Differenzspektren.py
5
6Liest tab-Spektren einer Serie ein und bildet Differenzspektren zum
7angegebenen mittleren Spektrum, die dann als tab-Datei ausgegeben werden.
8Alle Differenzspektren werden in einem Plot grafisch dargestellt, der auch
9abgespeichert wird.
10
11Created on Sat Feb 27 23:15:12 2021
12
13@author: lothar
14"""
15import numpy as np
16from astropy.io import ascii
17import matplotlib.pyplot as plt
18import glob
19
20
21# Fileliste erstellen. Spektren in einem (Unter)Ordner.
22files = input("Pfad und Name der Spektren (nutze wildcards) : ")
23filelist = glob.glob(files)
24
25# Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
26# zeitliche Ordnung
27filelist.sort()
28
29mean_name = input("Name Mittleres Spektrum: ")
30
31# Ausdruck der Liste
32print("\nSpektrenliste: \n")
33print(filelist)
34print("Anzahl der Spektren: ", len(filelist), "\n")
35print("Mittleres Spektrum", mean_name)
36
37
38mean_spec = ascii.read(mean_name, format="tab")
39
40fig = plt.figure(figsize=(14, 14))
41plt.xlabel("Wellenlänge [Angstrom]")
42plt.ylabel("Differenz Flux")
43
44for i in range(len(filelist)):
45    spec = ascii.read(filelist[i], format="tab")
46    diff = np.zeros(len(spec["WAVE"]))
47    for j in range(len(spec["FLUX"])):
48        diff[j] = spec["FLUX"][j] - mean_spec["FLUX"][j]
49    filename = filelist[i].rsplit(".")[-1] + "_diffspektrum" + ".dat"
50    ascii.write(
51        [spec["WAVE"], diff[:]],
52        filename,
53        overwrite=True,
54        names=["WAVE", "FLUX"],
55        format="tab",
56    )
57    plt.plot(spec["WAVE"], diff)
58
59fig.savefig("Differenzspektren" + "overplot.pdf")

```

Figure 16.1: Skript zur Bildung von Differenzspektren



```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Einlesen eines 1d-Spektrums in Form einer fits-Datei.
5
6Das Spektrum kann mit einer Standardabweichung, ausgedrückt durch die Auflösung
7R, convolviert werden.
8Die convolvierte fits-Datei wird abgespeichert.
9
10Stand 20220105
11@author: Lothar Schanne
12"""
13# from __future__ import print function, division
14import matplotlib.pyplot as plt
15from astropy.convolution import Gaussian1DKernel
16from astropy.convolution import convolve
17from astropy.io import ascii, fits
18
19spectrum_name = input('Pfad und Filebezeichnung eingeben: ')
20R = float(input('Gewünschtes R: '))
21
22# Spektrum einlesen
23sp = fits.open(spectrum_name)
24# print('\n\nHeader of the spectrum :\n\n', sp[0].header, '\n\n')
25std= sp[0].header['CRVAL1']/R/sp[0].header['CDELTA1']
26
27# Convolve mit astropy.convolution
28kernel = Gaussian1DKernel(stddev=std)
29sp[0].data = convolve(sp[0].data, kernel, normalize_kernel=True,
30                      boundary='extend')
31
32name = spectrum_name.rstrip('.fits')+'_convolved_R'+str(R)+'.fits'
33
34
35# Speichern des convolved
36fits.writeto(name, sp[0].data, sp[0].header, overwrite=True,
37            output_verify='ignore')
```

Figure 17.1: Skript zur Reduzierung der Auflösung eines 1d-Spektrums im fits-Format auf einen bestimmten Wert R.

```

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe
Öffnen [F4] Speichern

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Einlesen eines 1d-Spektrums in Form einer ascii-Tabelle mit extension .dat,
2-spaltig mit den Spaltenüberschriften 'WAVE' und 'FLUX'.
Das Spektrum kann mit einer Standardabweichung convolviert werden
(stddev in der Einheit des step).
Der convolvierte Flux wird zusammen mit der Wellenlänge in einer zweiseitigen
ascii-Tabelle abgespeichert (spacer = tab). Mit Spaltenüberschriften
'WAVE' und 'FLUX'.

Stand 20180824
@author: Lothar Schanne
"""
# from __future__ import print_function, division
import matplotlib.pyplot as plt
from astropy.convolution import Gaussian1DKernel
from astropy.convolution import convolve
from astropy.io import ascii

spectrum_name = input('Pfad und Filebezeichnung eingeben: ')

std = float(input('Standardabweichung für die Convolution in Vielfachen der\
Schrittweite: '))

# Spektrum einlesen
spectrum = ascii.read(spectrum_name, guess=True)

# Convolve mit astropy.convolution
kernel = Gaussian1DKernel(stddev=std)
convoluted = convolve(spectrum['FLUX'], kernel)

name = spectrum_name+'_convolved.dat'
# Grafik
fig = plt.figure(1, figsize=(14, 10))
plt.suptitle('Spektrum '+spectrum_name)
plt.subplot(2, 1, 1)
plt.plot(spectrum['WAVE'], spectrum['FLUX'])
plt.xlabel('Wellenlänge in Angström')
plt.ylabel('relative Flux')
plt.subplot(2, 1, 2)
plt.plot(spectrum['WAVE'], convoluted)
plt.xlabel('Wellenlänge in Angström')
plt.ylabel('relative Flux')
plt.title(name)
# speichern
plt.savefig(name+'.png')
plt.savefig(name+'.pdf')
# Speichern des convoluted
ascii.write([spectrum['WAVE'], convoluted], name, overwrite=True,
names=['WAVE', 'FLUX'], format='tab')

```

Python Tabulatorbreite: 4 Z. 8, Sp. 23 EINF

Figure 18.1: Das Skript *convol_dat.py* zum Falten eines Spektrums mit einer beliebigen Gaussfunktion.

der Schrittweite des Spektrums). Das gefaltete Spektrum wird als tab-separierte float-Zahlenpaare in zwei Spalten 'WAVE' und 'FLUX' abgespeichert. Der ursprüngliche Dateinamen wird zur Unterscheidung vom Original um den Anhang „_convoluted“ ergänzt.

19. Modifizierung eines theoretischen Spektrums aus der Pollux-Datenbank auf eine bestimmte Schrittweite und Auflösung

Nachfolgend wird ein Python3-Skript *Pollux_Ausschnitt_rebinned_convol_spec.py* vorgestellt (Programmlisting in 19.1), welches Modellspektren aus der Pollux-Datenbank (<http://pollux.gaal.univ-montp2.fr/>) verarbeitet. Es wurde mit den physikalischen Parametern von del Cep (6000 K, logg 2.0) ein Pollux-Spektrum aus der Polluxdatenbank erzeugt und im spec-Format heruntergeladen. Dabei werden 2 ascii-Dateien dem Nutzer übergeben: xyz.txt, welche die Berechnungsparameter enthält sowie eine xyz.spec-Datei, die 3 Spalten ohne Überschrift enthält: Wellenlänge, absoluter Flux und normierter Flux.

Nach dem Start des Programms werden nacheinander abgefragt: Pfad und Bezeichner des Spektrums, Beginn und Ende des gewünschten Wellenlängenausschnitts, die FWHM des Apparateprofils und die Schrittweite des zu erzeugenden Spektrums. Es wird dann eine Grafik ausgegeben mit dem rebinnten

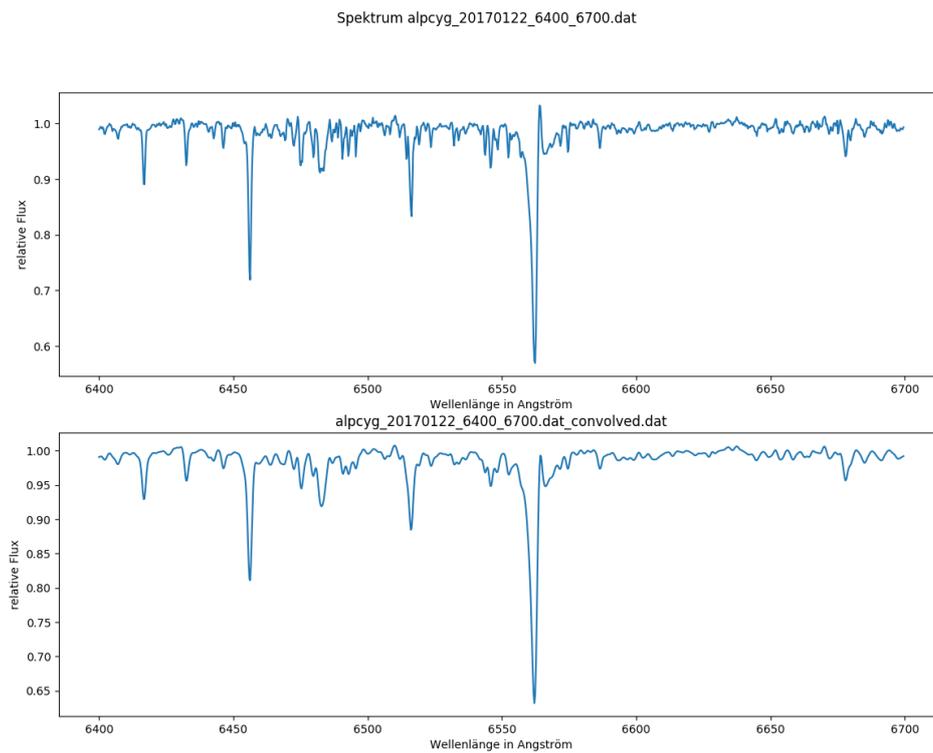


Figure 18.2: Ergebnisgrafik des Skripts `convol_dat.py` zum Falten eines Spektrums mit einer beliebigen Gaussfunktion. Hier wurde mit dem Parameter $std = 2$ (\approx Verdoppelung des gerade noch aufgelösten Elements, Halbierung der relativen Auflösung R) gearbeitet. Die Breite der Linien wurde etwa verdoppelt.

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3 """
4 Einlesen eines synthetischen Spektrums in Form einer Tabelle (wie als .spec
5 in der Pollux-Datenbank erhältlich).
6 Berechnung eines wählbaren Wellenlängenausschnitts (Pandas Dataframe mit
7 'newtable' bezeichnet). Dieser Bereich wird dann mit einer wählbaren
8 FWHM (Apparaturprofil) gefaltet.
9 Geplottet wird der gewählte rebinnete Wellenlängenbereich und zusätzlich das
10 gefaltete Spektrum.
11 Der rebinnete Flux-Ausschnitt des ursprünglichen .spec und der
12 rebinnete und convolierte Flux wird zusammen mit der Wellenlänge in je einer
13 zweiseitigen ascii-Tabelle (spacer = tab) und in einer fits-Datei
14 abgespeichert.
15
16 Stand 20220129
17 @author: lothar
18 """
19 # from __future__ import print_function, division
20 import numpy as np
21 import matplotlib.pyplot as plt
22 from astropy.convolution import Gaussian1DKernel, convolve
23 from astropy.io import ascii, fits
24 from PyAstronomy.pyasl import binningx0dt
25 import pandas as pd
26
27 file = input("Pfad und Datei bezeichnung eingeben: ")
28 table = pd.read_fwf(file, names=["WAVE", "AFUX", "NFLUX"], header=0)
29 begin = float(
30     input("Geben Sie den Beginn des gewünschten Wellenlängenbereichs in
31     Angström ein: ")
32 )
33 end = float(input("Geben Sie das Ende des gewünschten Wellenlängenbereichs
34 ein: "))
35 fwhm = float(
36     input("Geben Sie die gewünschte FWHM des Apparaturprofils in Angström ein: ")
37 )
38 step = float(input("Geben Sie die gewünschte Schrittweite des
39 Ergebnisspektrums ein: "))
40 newtable = table[table["WAVE"] >= begin]
41 newtable = newtable[newtable["WAVE"] <= end]
42 # Binning
43 data, d = binningx0dt(
44     newtable["WAVE"],
45     newtable["NFLUX"],
46     dt=step,
47     x0=newtable["WAVE"].min(),

```

```

47     x0=newtable["WAVE"].min(),
48     removeEmpty=False,
49 )
50 data = data.T
51 wave = data[0]
52 flux = data[1]
53
54 # Convolve mit astropy.convolution
55 kernel = Gaussian1DKernel(stddev=fwhm / 2.3 / step)
56 convoluted = convolve(flux, kernel, normalize_kernel=True, boundary="extend")
57
58 # Grafik
59 plt.style.use('seaborn-whitegrid')
60 fig, ax = plt.subplots(2)
61 plt.xlabel("Wellenlänge [Angström]", fontsize=5)
62 fig.suptitle("Pollux-Spektrum " + file, fontsize=5)
63 ax[0].plot(wave, flux, linewidth=0.1)
64 # ax[0].set_xlim(4000, 6700)
65 # ax[0].set_ylim(0.2, 1.1)
66 ax[0].tick_params(axis="both", labelsize=3)
67 ax[0].grid(b=True, axis="both", linewidth=0.2)
68 ax[1].plot(wave, convoluted, linewidth=0.2)
69 ax[1].tick_params(axis="both", labelsize=3)
70 # ax[1].set_xlim(4000, 6700)
71 # ax[1].set_ylim(0.2, 1.1)
72 ax[1].grid(b=True, axis="both", linewidth=0.2)
73 name = file.rstrip(".spec")
74 plt.savefig(file+'.png')
75 plt.savefig(name + ".pdf")
76
77 # Speichern des beschnittenen .spec, NFLUX jetzt als FLUX als dat und fits
78
79 # nicht convolviert
80 name = (
81     file.rstrip(".spec") + "_" + str(int(begin)) + "_" + str(int(end)) +
82     "_rebinned.dat"
83 )
84
85 name = (
86     file.rstrip(".spec")
87     + "_"
88     + str(int(begin))
89     + "_"
90     + str(int(end))
91     + "_rebinned.fits"
92 )
93
94 header = fits.Header()
95 header["SIMPLE"] = "T"
96 header["BITPIX"] = -32

```

```

95 header["SIMPLE"] = "T"
96 header["BITPIX"] = -32
97 header["NAXIS"] = 1
98 header["CRVAL1"] = wave[0]
99 header["NAXIS1"] = len(wave)
100 header["CDELT1"] = step
101 header["CUNIT1"] = "Angstrom"
102 header["CTYPE1"] = "Wavelength"
103 header["CRPIX1"] = 1
104
105 fits.writeto(
106     name, flux, header, overwrite=True, output_verify="ignore",
107 )
108
109 # convolviert
110 name = (
111     file.rstrip(".spec")
112     + "_"
113     + str(int(begin))
114     + "_"
115     + str(int(end))
116     + "_rebinned_convolved.dat"
117 )
118
119 ascii.write(
120     [wave, convoluted], name, overwrite=True, names=["WAVE", "FLUX"],
121     format="tab"
122 )
123 name = (
124     file.rstrip(".spec")
125     + "_"
126     + str(int(begin))
127     + "_"
128     + str(int(end))
129     + "_rebinned_convolved.fits"
130 )
131
132 header = fits.Header()
133 header["SIMPLE"] = "T"
134 header["BITPIX"] = -32
135 header["NAXIS"] = 1
136 header["CRVAL1"] = wave[0]
137 header["NAXIS1"] = len(wave)
138 header["CDELT1"] = step
139 header["CUNIT1"] = "Angstrom"
140 header["CTYPE1"] = "Wavelength"
141 header["CRPIX1"] = 1
142
143 fits.writeto(
144     name, convoluted, header, overwrite=True, output_verify="ignore",
145 )

```

Figure 19.1: Programmlisting des Skripts *Pollux_Ausschnitt_rebinned_convол_spec.py*.

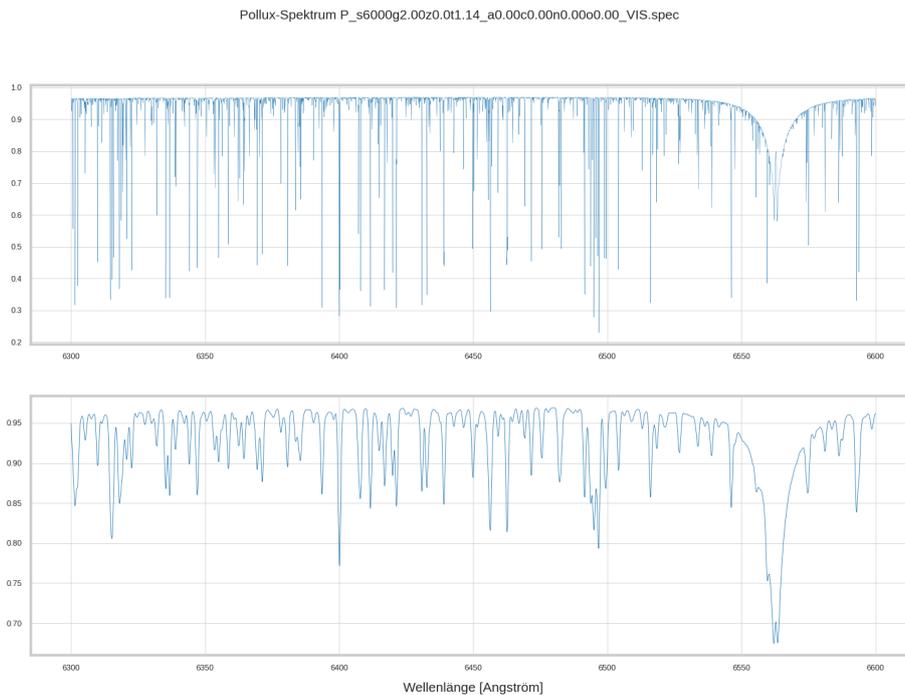


Figure 19.2: Grafik des Skripts *Pollux_Ausschnitt_rebinned_convolve_spec.py*, oben der rebinnete Wellenlängenausschnitt des gesamten Spektrums aus der Pollux-Datenbank, unten der zusätzlich mit dem Apparateprofil (hier 1 Angström) convolierte Ausschnitt.

Spektrumausschnitt und darunter der rebinnete und convolierte Ausschnitt. Abgespeichert werden die Dateien des rebinneten und des zusätzlich convolierten Wellenlängenausschnitts im ascii- und im fits-Format sowie die Grafik im PNG und im PDF-Format. In 19.2 ist die Grafik für das hier durchgeführte Beispiel gezeigt.

20. Bezug eines theoretischen Spektrums aus einer Datenbank von PyAstronomy mit einer bestimmten Auflösung R

In PyAstronomy ist eine Datenbank von Kurucz-Modellen implementiert. Während in der Pollux-Datenbank für den Aufruf eines theoretischen Spektrums viele physikalischen Kenngrößen eines Sterns erforderlich sind, ist in dieser Datenbank lediglich die Eingabe der wichtigsten Sterneigenschaften T_{eff} und $\log g$ erforderlich, was für viele Zwecke genügt.

Der Aufruf und die Umrechnung auf eine bestimmte gewünschte Auflösung R (z.B. 10000) erledigt das Pythonskript *Kurucz_Modell_Gaussbroadendes.py* (Pythoncode vgl. Abb. 20.1). Das Skript fragt nacheinander ab: T_{eff} , $\log g$, Bezeichner für das abzuspeichernde Spektrum, R , Schrittweite, Wellenlängenbereich.

21. Vergleich von theoretischem und gemessenen Spektrum (overplot)

Ist das nach den vorstehenden Abschnitten gefaltete und damit auf die Dispersion des gemessenen Spektrums angepasste theoretisch berechnete Modellspektrum auf der Festplatte kann es anschließend mit dem gemessenen grafisch verglichen werden. Das erledigt das Python3-Skript *Vergleichsplot_Modell_Messung.py* das in Abb. 21.1 gelistet ist. Es plottet zwei Spektren, die als tab-separierte ASCII-Datei vorliegen, übereinander und speichert den plot als PNG und PDF ab.

Ein Beispiel für das erzeugte Diagramm vgl. Abb. 21.2.

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Laden eines Kurucz-Sternatmosphärenmodells (1d-Spektrum) nach Auswahl von Teff
5und logg. Konvolution mit einer Gaußfunktion auf Spektrogrammauflösung.
6Auswahl des Wellenlängenbereiches und
7Anpassung auf eine gewünschte Schrittweite. Speicherung als ascii-tab-Tabelle
8und als .fit mit dem einzuübenden Filenamen. Speicherung des plot als pdf.
9
10Created on Fri Apr 2 13:53:54 2021
11
12@author: lothar
13"""
14from PyAstronomy import pyasl
15import matplotlib.pyplot as plt
16from astropy.io import ascii
17from astropy.io import fits
18
19# Auswahl des Modells mittels Teff, logg und Metallizität
20model = pyasl.resBased.spectralLib.SpectralLib(lib='A1')
21print('Liste der zur Verfügung stehenden Modelle (Parameter und Filenamen):')
22model.listInventory()
23
24Teff = float(input('Geben Sie Teff ein: '))
25logg = float(input('Geben Sie logg ein: '))
26met = 0.0 # momentan gibt es nur Modelle mit Metallicity = 0.0
27# met = float(input('Geben Sie die Metallizität ein: '))
28
29Model = model.requestModel(Teff, logg, met)
30wave, flux = model.readIdFitsSpec(Model)
31
32specname = input('Dateibezeichnung für Speicherung eingeben: ')+'.fit'
33
34print('Schrittweite des Modellspektrums = ', wave[1]-wave[0])
35R = float(input('Gewünschte relative Auflösung R eingeben (z.B. 10000): '))
36newstep = float(
37    input('Geben Sie die gewünschte Schrittweite des Endergebnisses ein: '))
38spectrum_name = specname + str(Teff) + '_' + str(logg) + '_R' + str(int(R))
39
40print('\nEingabe des gewünschten Wellenlängenbereiches in Angström:')
41a = float(input('Begin: '))
42b = float(input('End: '))
43
44for m in range(len(wave)):
45    if wave[m] > a:
46        aind = m
47        break
48for m in range(len(wave)):
49    if wave[m] > b:
50        bind = m
51        break
52wave = wave[aind:bind]
53flux = flux[aind:bind]
54
55
56convolutedFlux, fwhm = pyasl.instrBroadGaussFast(
57    wave, flux, R, edgeHandling='firstlast', fullout=True, maxsig=5.0)
58print('FWHM des gaußverbreiterten Spektrums = ', fwhm)
59
60data, dt = pyasl.binningx0dt(
61    wave, convolutedFlux, x0=wave[0], dt=newstep, useBinCenter=True, useMean=True)
62name = spectrum_name + '_convolved.dat'
63ascii.write([data[:, 0], data[:, 1]], name, overwrite=True,
64    names=['WAVE', 'FLUX'], format='tab')
65hdu = fits.PrimaryHDU()
66hdu.header['CRVAL1'] = data[0, 0]
67hdu.header['CRPIX1'] = 1
68hdu.header['NAXIS1'] = len(data)
69hdu.header['CDELT1'] = newstep
70hdu.header['NAXIS'] = 1
71name = spectrum_name + '_convolved.fit'
72fits.writeto(name, data[:, 1], hdu.header, overwrite=True)
73
74
75# Grafik
76fig, ax = plt.subplots(nrows=2, ncols=1)
77fig.suptitle('Spektrum: ' + spectrum_name, fontsize=12)
78
79ax[0].plot(wave, flux, linewidth=3)
80ax[0].set_ylabel('relative Flux')
81ax[0].set_title('Kurucz-Modell', fontsize=8)
82
83ax[1].plot(data[:, 0], data[:, 1], linewidth=3)
84ax[1].set_xlabel('Wellenlänge in Angström', ylabel='relative Flux')
85ax[1].set_title('convolved', fontsize=8)
86ax[1].set_xlim(6500, 6700)
87
88plt.savefig(spectrum_name + '_convolved.pdf')

```

Figure 20.1: Code des Skripts Kurucz_Modell_Gaussbroadendes.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Skript liest ein mit 'Pollux_Ausschnitt.py' erzeugter ascii-file ein und plottet
ihn.
Das zu vergleichende gemessene Spektrum wird gewählt und overplottet.

Stand 20180824
@author: lothar
"""

import matplotlib.pyplot as plt
from astropy.io import ascii
# from astropy.io import fits
# import numpy as np
# import pandas as pd

# Eingabe des in ascii, tab-splitted codierten Template-files
# Dieses muss 2 tab-separierte Spalten WAVE und NFLUX haben
template_name = input('Geben Sie Pfad und Namen des convolvierten Templates\
ein: ')
spectrum_name = input('Geben Sie Pfad und Namen des gemessenen\
ein: ')

template = ascii.read(template_name, format='tab')
spectrum = ascii.read(spectrum_name, format='tab')

fig = plt.figure(figsize=(14, 14))
plt.plot(template['WAVE'], template['FLUX'])
plt.plot(spectrum['WAVE'], spectrum['FLUX'])
plt.xlabel('Wellenlänge in Angström')
plt.ylabel('relative Flux')
plt.title(spectrum_name.rstrip('.dat'))
plt.legend([template_name.rstrip('.dat'), spectrum_name.rstrip('.dat')],
    loc='lower left')

fig.savefig(spectrum_name.rstrip('.dat')+'templateVergleich.png')
fig.savefig(spectrum_name.rstrip('.dat')+'templateVergleich.pdf')

```

Figure 21.1: Skript-Listing von Vergleichsplot_Modell_Messung.py.

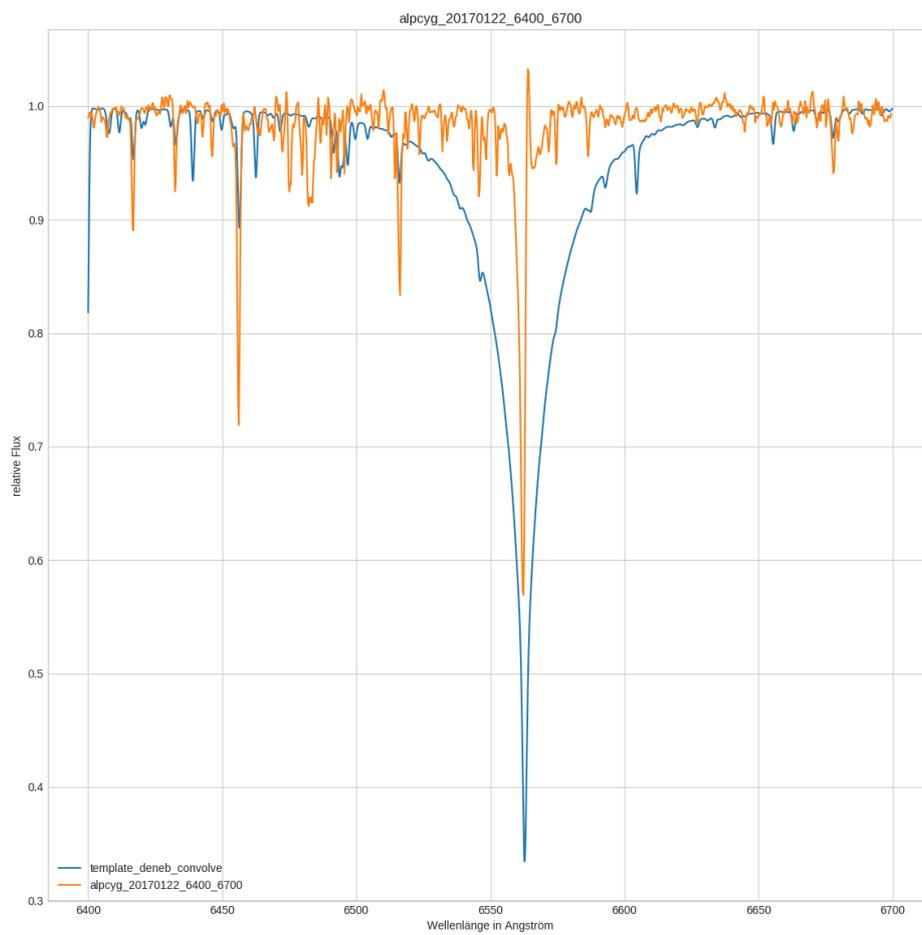


Figure 21.2: Von *Vergleichsplot_Modell_Messung.py* erzeugte Grafik.

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Normierung eines fits-Spektrums mittels interaktiv festgelegter Stützpunkte.
5Grafische Darstellung des unnormierten Spektrums, der festgelegten Stützstellen
6und des Splines durch die Punkte. Abspeicherung der Stützpunkte in einer Datei
7namens Wellenlaengen_Stuetzpunkte.npy.
8
9Created on Sat Oct 10 13:54:05 2020
10
11@author: lothar
12"""
13
14import numpy as np
15import matplotlib.pyplot as plt
16from astropy.io import fits
17from scipy.interpolate import UnivariateSpline
18
19#####
20# Anderbare Konstanten, jeweils anzupassen an die Liniendichte und Linienbreite
21
22# Smoothing Faktor für die Spline-Übertragung
23sm = 0.001
24
25#####
26
27# adjust the path and name of the spectrum file.
28file = input('Path and name of the fits-file: ')
29
30
31# Header-Check Spectrum
32sp = fits.open(file)
33# print('\n\nHeader of the spectrum :\n\n', sp[0].header, '\n\n')
34
35# Generation of arrays with the wavelengths and fluxes of the spectrum
36flux = np.array(sp[0].data)
37wave = np.ones(sp[0].header['NAXIS1'], dtype=float)
38sp[0].header['CRVAL1'] = sp[0].header['CRVAL1'] + \
39    (1 - sp[0].header['CRPIX1']) * sp[0].header['CDELTA1']
40for i in range(sp[0].header['NAXIS1']):
41    wave[i] = sp[0].header['CRVAL1'] + i * sp[0].header['CDELTA1']
42# The list wave contains the wavelengths of the pixels.
43# In the list flux the corresponding intensities.
44
45# Close the fits-file
46sp.close()
47
48
49# Plot the spectrum
50fig = plt.figure(1, figsize=(14, 10))
51plt.plot(wave, flux)
52plt.xlabel('Wavelength [Angström]')
53plt.ylabel('ADU')
54plt.title('Spektrum '+file)
55plt.grid(True)
56
57
58# Interaktives Festlegen der Stützpunkte zum Normieren durch linksMaus-Klicks
59# Zum Beenden der Interaktivität zweimal die ESC Taste drücken
60# plt.waitforbuttonpress()
61while True:
62    pts = []
63    pts = np.asarray(plt.ginput(n=-1, timeout=-1))
64    if plt.waitforbuttonpress():
65        break
66
67plt.plot(pts[:, 0], pts[:, 1], 'o', markersize=6)
68
69
70# Erzeugung eines kubischen splines aus den Stützpunkten
71
72spl = UnivariateSpline(pts[:, 0], pts[:, 1])
73spl.set_smoothing_factor(sm)
74# Spline angewendet auf die Wellenlängen des spectrum
75normfunction = spl(wave)
76normspectrumflux = flux / normfunction
77plt.plot(wave, normfunction)
78
79# abspeichern des normierten fit
80filename = file.rsplit('.', 1)
81filename = filename[0] + '_normiert.fit'
82sp[0].header['HISTORY'] = 'Normalization by Normierung_Spline_interaktiv.py'
83fits.writeto(filename, normspectrumflux, sp[0].header, overwrite=True)
84
85# Plot the spectrum
86fig = plt.figure(2, figsize=(14, 10))
87plt.plot(wave, normspectrumflux)
88plt.xlabel('Wavelength [Angström]')
89plt.ylabel('ADU')
90plt.title(filename)
91plt.grid(True)
92plt.show()
93
94# Abspeichern der Stützpunkte-Wellenlängen als Binärdatei
95with open('Wellenlaengen_Stuetzpunkte.npy', 'wb') as f:
96    np.save(f, pts[:, 0])

```

Figure 22.1: Pythonskript zur interaktiven Normierung eines 1d-Spektrums im fits-Format.

Part VI.

Normierung von 1d-Spektren

Das Normieren von 1d-Spektren auf das Kontinuum ist eine häufig zu erledigende Aufgabe. Die meisten astronomischen Spektroskopieprogramme können dies mit einzelnen fits-Dateien. Eine Alternative dazu bilden die folgenden Pythonskripte, mit denen einzelne oder ganze Serien von 1d-Spektren auf einmal auf das Kontinuum normiert werden.

22. Interaktive Festlegung von Normierungsstützstellen und Normierung mittels eines Splines

Das hier besprochene Skript erlaubt die grafische Festlegung von Normierungsstützstellen durch Mausklicks. Es werden also linienfreie Stellen (= angenommenes Kontinuum) eines nicht normierten Spektrums angeklickt. Deren Wellenlänge wird für die Normierung weiterer Spektren des gleichen Objekts registriert und abgespeichert. Durch die gewählten Stützstellen wird ein kubischer Spline gelegt, dessen Rigidität mittels eines im Skript anpassbaren „Smoothingfaktors“ *sm* gesteuert wird und mittels dieses Splines das Spektrum normiert.

23. Normierung einer Serie von 1d-Spektren mittels einer Liste festgelegter Wellenlängen der Normierungsstützstellen

Liegt eine Serie von Spektren des gleichen Wellenlängenbereiches und gleichen Objektes zum Normieren vor, kann die im vorhergehenden Skript an einem Spektrum erzeugte Liste der Normierungsstützstellen verwendet werden. Das erledigt das Skript „Normierung_Zeitreihe_Spline_festgelegtePunkte.py“. Man erhält eine Serie von 1d-Spektren, die alle mit den gleichen Stützstellen(wellenlängen) normiert

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sat Oct 10 14:32:30 2020
5Liest eine Zeitserie von Spektren im fits-Format ein und normiert sie an Hand
6einer mit dem Programm Normierung_Spline_Interaktiv.py festgelegten Stützpunkte
7(Datei 'Wellenlaengen_Stuetzpunkte.npy'), speichert sie mit dem Namenszusatz
8'normiert' als fit ab.
9Mit dem Parameter sm wird die Übertragung des Splines auf die Wellenlängen
10optimiert.
11Das Intervall gibt die Halbe Breite des Intervalls (in Pixel) an, in dem um
12die Stützpunktwellenlänge die flux-Werte mit sigma-clipping gemittelt werden
13(median).
14
15@author: lothar
16"""
17
18import numpy as np
19import matplotlib.pyplot as plt
20from astropy.io import fits
21from scipy.interpolate import UnivariateSpline
22from scipy.stats import sigmaclip
23import glob
24
25#####
26# Anderbare Konstanten, jeweils anzupassen an die Liniendichte und Linienbreite
27
28# Smoothing Faktor für die Spline-Übertragung
29sm = 0.001
30intervall = 3
31
32#####
33
34# Laden der Wellenlängen der Normierungs-Stützpunkte
35WL = np.load("Wellenlaengen_Stuetzpunkte.npy")
36
37# Create file list. All spectra in one (sub)folder.
38files = input("Path and name of the file (use wildcards) : ")
39filelist = glob.glob(files)
40
41# Alphabetical sorting. If the name is correct, this results in a
42# temporal order
43filelist.sort()
44
45# Print the list
46print(filelist)
47
48# Print the list
49print("\nLIST of spectra: \n")
50print(filelist)
51print("Number of spectra: ", len(filelist), "\n")
52
53for n in range(len(filelist)):
54    sp = fits.open(filelist[n])
55    # Generation of arrays with the wavelengths and fluxes of the spectrum
56    flux = np.array(sp[0].data)
57    wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
58    for i in range(sp[0].header["NAXIS1"]):
59        wave[i] = sp[0].header["CRVAL1"] + i * sp[0].header["CDELTA1"]
60    # The list wave contains the wavelengths of the pixels.
61    # In the list flux the corresponding intensities.
62
63    # Close the fits-file
64    sp.close()
65
66    # Suchen der Wellenlängen und Flux für die Stützpunkte im Spectrum
67    wave_stuetz = np.zeros(len(WL))
68    flux_stuetz = np.zeros(len(WL))
69    for i in range(len(WL)):
70        for j in range(len(wave)):
71            if wave[j] >= WL[i]:
72                wave_stuetz[i] = wave[j]
73                inter = np.zeros(2 * intervall)
74                inter = flux[j - intervall : j + intervall]
75                inter_low, up = sigmaclip(inter, 3, 3)
76                flux_stuetz[i] = np.median(inter)
77                break
78
79    # Erzeugung eines kubischen splines aus den Stützpunkten
80    spl = UnivariateSpline(wave_stuetz, flux_stuetz)
81    spl.set_smoothing_factor(sm)
82    # Spline angewendet auf die Wellenlängen des spectrum
83    normfunction = spl(wave)
84    normspectrumflux = flux / normfunction
85    # abspeichern des normierten fit
86    filename = filelist[n].rsplit('.', 1)
87    filename = filename[0] + "_normiert.fits"
88    sp[0].header["HISTORY"]
89    ] = "Normalization by Normierung_Zeitserie_Spline_festgelegtePunkte.py"
90    fits.writeto(filename, normspectrumflux, sp[0].header, overwrite=True)

```

Figure 23.1: Skript zur Normierung einer Serie von 1d-Spektren im fits-Format.

wurden.

24. Allgemeine Routine zur Normierung ganzer Spektrenserien im fits-Format

Die Definition des Kontinuumverlaufs zur Normierung der Spektren kann in vielen Spektren sehr schwierig sein:

1. Manche Absorptionslinien sind sehr breit (z.B. die H alpha Linie bei 6563 Å), so daß über einen breiten Wellenlängenbereich keine Normierungsstützpunkte zu definieren sind.
2. Spektren später Sterne sind sehr linienreich. Es existieren - wenn überhaupt - nur schmale Wellenlängenbereiche, die als Kontinuum identifiziert werden können.
3. Aus apparativen Gründen können die Kontinua der Spektren sehr wellig sein (z.B. gemergte Echellespektren mit großem Wellenlängenbereich).

Es wurde deshalb ein umfangreiches Pythonskript entwickelt, das möglichst universell automatisch das Quasikontinuum findet und Steuerungsmöglichkeiten enthält, welche die obig aufgeführten Schwierigkeiten befriedigend umschiffen können. Da im Falle größerer Spektrenserien viel Rechenleistung erfordert wird, werden alle bis auf einen Prozessorkerne des PC's verwendet.

Wir stellen das Skript nachfolgend in einzelnen Schritten vor (Abb. 24.1 bis 24.5). Es kann wie alle anderen in diesem Buch beschriebenen Pythonskripte von der Webseite xxx heruntergeladen werden. Damit das Programm läuft müssen natürlich alle mit den import-Befehlen (Zeilen 22 bis 31) angesprochenen Module im Pythonsystem installiert sein.

Nach der Definition einiger Funktionen, die im Programmablauf mehrfach verwendet werden, werden ab Zeile 180 am ersten Spektrum der Serie die Parameter optimiert. Dazu wird das Spektrum geplottet (Zeile 194), lokale Maxima berechnet (203) und die so ermittelten vorläufigen Normierungsstützstellen geplottet (205). Falls es zu viele sind oder zu viele innerhalb von Linien zu finden sind, kann das Maxima-Suchintervall „inter“ beliebig oft verändert/erhöht werden. Das neue Ergebnis an Stützstellen wird jeweils geplottet.

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3
4 ---
5 Works for a single or a time series of 1d spectra in fits format.
6 At first spectrum the parameters 'inter' and 'sm' are optimized.
7 Takes from pixel to pixel an interval of 'inter' pixels, calculated
8 of which is a percentile and compares it with the neighboring intervals.
9 Finds so local maxima = support points.
10 You can also distinguish several wavelength ranges from the formation of
11 normalization points, i.e. delete them (wide lines).
12 The Smoothing parameter 'sm' adjusts the sensitivity of the spline
13 which is used for normalization at the end. After that, you can still delete
14 points that are a certain percentage larger or smaller than the spline.
15 Finally, all spectra of the time series are normalized with the optimized
16 parameters.
17
18 07.12.2020
19 @author: lothar
20 ***
21
22 import numpy as np
23 import matplotlib.pyplot as plt
24 from astropy.io import fits
25 from scipy.stats import sigmaclip
26 import glob
27 import concurrent.futures as futures
28 import os
29 from csaps import csaps
30
31 # from numba import jit
32
33
34 # definition of functions
35
36
37 def wavecalc(flux, header):
38     flux_median = np.nanmedian(flux)
39     wave = np.zeros(header["NAXIS1"])
40     if "CRPIX1" not in header:
41         header["CRPIX1"] = 1
42     header["CRVAL1"] = header["CRVAL1"] + (1 - header["CRPIX1"]) *
43     header["CDELTA1"]
44     for i in np.arange(header["NAXIS1"]):
45         wave[i] = header["CRVAL1"] + i * header["CDELTA1"]
46     wave = wave[flux >= flux_median * 0.001]
47     flux = flux[flux >= flux_median * 0.001]
48     return wave, flux

```

```

49
50 def maximacalc(inter, flux, wave):
51     pixelanzahl = len(wave)
52     maximas_flux = np.zeros(pixelanzahl)
53     maximas_wave = np.zeros(pixelanzahl)
54     points_flux = np.zeros(pixelanzahl)
55     points_wave = np.zeros(pixelanzahl)
56     for n in np.arange(inter, pixelanzahl - 2 * inter):
57         interflux = flux[n : n + inter] # Flux des Intervalls n
58         inter_percentil = np.percentile(interflux, 50)
59         b = np.percentile(flux[n - inter : n], 75)
60         c = np.percentile(flux[n + inter : n + 2 * inter], 75)
61         if b < inter_percentil > c:
62             interflux1, low, up = sigmaclip(interflux, 2, 2)
63             maximas_flux[n + inter // 2] = np.percentile(interflux1, 50)
64             maximas_wave[n + inter // 2] = wave[n + inter // 2]
65     points_flux = maximas_flux[maximas_flux != 0]
66     points_wave = maximas_wave[maximas_wave != 0]
67     return points_flux, points_wave
68
69
70 # @jit
71 def reduction(s1, s2, points_wave, points_flux, wave, normfunction):
72     for n in np.arange(len(points_wave)):
73         for i in np.arange(len(wave)):
74             if wave[i] == points_wave[n] and (
75                 points_flux[n] < (1 - s1 / 100) * normfunction[i]
76                 or points_flux[n] > (1 + s2 / 100) * normfunction[i]
77             ):
78                 points_flux[n] = 0
79                 break
80     return points_flux
81
82
83
84
85 # @jit
86 def cutting(wave, points_wave):
87     for n in np.arange(len(wave)):
88         if wave[n] < points_wave[0]:
89             wave[n] = 0
90         if wave[n] > points_wave[-1]:
91             wave[n] = 0
92     return wave
93
94
95 def pipe(kl):
96     flux, header = fits.getdata(filelist[kl], header=True)
97     # Generation of arrays with the wavelengths and fluxes of the spectrum
98     wave, flux = wavecalc(flux, header)

```

Figure 24.1: Pythonskript zum automatischen Normieren von Spektren im fits-Format.

```

96     flux, header = fits.getdata(filelist[kl], header=True)
97     # Generation of arrays with the wavelengths and fluxes of the spectrum
98     wave, flux = wavecalc(flux, header)
99
100     points_flux, points_wave = maximacalc(inter, flux, wave)
101
102     for i in np.arange(len(loeschl)):
103         for n in np.arange(len(points_wave)):
104             if loeschl[i] < points_wave[n] and loeschl2[i] > points_wave[n]:
105                 points_flux[n] = 0
106
107     points_wave = points_wave[points_flux != 0]
108     points_flux = points_flux[points_flux != 0]
109
110     # Restriction of the wavelength range to the range of the interpolation
111     points
112     wave = cutting(wave, points_wave)
113     flux = flux[wave != 0]
114     wave = wave[wave != 0]
115
116     # Creation of a cubic spline from the points (= normfunction)
117     normfunction = csaps(points_wave[:,], points_flux[:,], wave[:,], smooth=sm)
118
119     if frage2[0] == "y":
120         for m in range(len(s1)):
121             points_flux = reduction(
122                 s1[m], s2[m], points_wave, points_flux, wave, normfunction
123             )
124             points_wave = points_wave[points_flux != 0]
125             points_flux = points_flux[points_flux != 0]
126             wave = cutting(wave, points_wave)
127             flux = flux[wave != 0]
128             wave = wave[wave != 0]
129             normfunction = csaps(points_wave[:,], points_flux[:,], wave[:,],
130 smooth=sm)
131
132     # Normalization
133     normspectrumflux = flux / normfunction
134     ausdruck = (
135         + str(kl + 1)
136         + " "
137         + filelist[kl]
138         + " " + str(len(points_wave))
139     )
140     print(ausdruck)
141
142

```

```

142
143     return (
144         kl,
145         normfunction,
146         normspectrumflux,
147         points_flux,
148         points_wave,
149         wave,
150         flux,
151         header,
152     )
153
154
155 #####
156 # Initial values of changeable constants, later adapted to the line density
157 # and line width and number of interpolation points of spline
158 inter = 10 # Intervall (Pixel)
159
160 sm = 0.001 # initial smoothing factor for spline
161
162 #####
163
164 # plt.ion()
165
166 # *****
167 # Create file list. Spectra in a (sub)folder.
168 files = input("Path and name of the spectra (use wildcards) : ")
169 filelist = glob.glob(files)
170
171 # Sort alphabetically. If the spectrum files are named correctly, this results
172 # in a temporal order.
173 filelist.sort()
174
175 # Printout of the list for control purposes.
176 print("\nList of spectra:")
177 print(filelist)
178 print("\nNumber of spectra: ", len(filelist), "\n")
179
180 # ***** Optimizing the parameters on the first spectrum
181 print(
182     "We first optimize the parameter 'interval width' and select wavelength
183     ranges in which no interpolation points are desired.\n"
184 )
185 k = 0
186 flux, header = fits.getdata(filelist[0], header=True)
187 print("\n\nHeader of the spectrum :\n\n", sp[0].header, "\n\n")
188
189 # Generation of arrays with the wavelengths and fluxes of the spectrum

```

Figure 24.2: Pythonskript zum automatischen Normieren von Spektren im fits-Format. Fortsetzung.

```

189 # Generation of arrays with the wavelengths and fluxes of the spectrum
190 wave, flux = wavelcalc(flux, header)
191 # The list wave contains the wavelengths of the pixels.
192 # In the list flux the corresponding intensities.
193
194 # Plot the spectrum
195 fig, ax = plt.subplots(1, figsize=(10, 8))
196 ax.plot(wave, flux)
197 ax.set_xlabel("Wavelength [Angström]")
198 ax.set_ylabel("ADU")
199 fig.suptitle("Spectrum " + filelist[0])
200 ax.grid(True)
201
202
203 points_flux, points_wave = maximacalc(inter, flux, wave)
204
205 # Plot of normalization points
206 ax.plot(points_wave, points_flux, "or", markersize=3)
207 plt.show(block=False)
208
209 # ***** Interval width adjusting
210 print("Current interval width \"inter\" is:", inter)
211 print("\nCurrent number of support points: ", len(points_wave))
212 forderung = input("Do you want to change the interval width \"inter\"? Then enter
y: ")
213 while forderung == "y":
214     inter = int(input("Enter the new interval width: "))
215     fig = plt.figure(figsize=(10, 8))
216     plt.plot(wave, flux)
217     plt.xlabel("Wavelength [Angström]")
218     plt.ylabel("ADU")
219     plt.title("Spectrum " + filelist[0])
220     plt.grid(True)
221
222     points_flux, points_wave = maximacalc(inter, flux, wave)
223
224     # Plot of normalization points
225     plt.plot(points_wave, points_flux, "or", markersize=3)
226     plt.show(block=False)
227
228     print("Current number of support points: ", len(points_wave))
229
230     forderung = input(
231         "Do you want to change the interval width \"inter\" again? Then enter y:
")
232
233
234
235 # ***** Exclude wavelength ranges
236 forderung = input(

```

Figure 24.3: Pythonskript zum automatischen Normieren von Spektren im fits-Format. Fortsetzung.

Anschließend (24.3) wird gefragt, ob man in bestimmten Wellenlängenbereichen keine Normierungsstützstellen bilden möchte. Das kann z.B. in einer breiten H alpha Linie der Fall sein. Man gibt dann den Beginn und das Ende des Wellenlängenbereiches ein, worauf die darin enthaltenen vorläufigen Stützstellen gelöscht werden. In der Konsole wird jeweils die Anzahl der verbliebenen Stützstellen gedruckt und das Spektrum und die aktuellen Stützstellen grafisch dargestellt. Danach wird durch die verbliebenen Stützstellen ein kubischer Spline gelegt (ab Zeile 275) und geplottet. Danach wird man gefragt, ob man den Smoothingfaktor verändern möchte. Mit dem Faktor kann man den Spline rigider machen, in dem man den Faktor „sm“ verkleinert, z.B. auf 0.001 oder 0.0001. Je näher er an Null liegt, desto steifer (rigider) wird der Spline und folgt nicht mehr so detailliert lokalen Schwankungen der Stützstellen. Man wählt den Smoothingfaktor so lange immer kleiner, bis der Spline dem gedachten Kontinuumverlauf möglichst deckungsgleich folgt.

Im nächsten und letzten Schritt entfernt man noch Stützstellen, die zu weit vom Spline abweichen, also zu weit darunter oder darüber liegen. Nach der Frage, ob man Stützstellen löschen möchte, die s1 % unter bzw. s2 % über dem Spline liegen, gibt man diese Grenzen ein, z.B. 0.5 für s1 und 5 für s2. Daraufhin werden alle Stützstellen gelöscht, die außerhalb dieser Grenzen um den Spline liegen. Das Spektrum, der neue Spline und der neue Satz von Stützstellen werden jeweils neu geplottet. Ist man zufrieden mit dem Ergebnis wird das Spektrum mit dem letzten Spline als Normierungsfunktion normiert (Zeile 349) und geplottet (368).

Nach weiteren Fragen über das Anzeigen von Graphiken und das Abspeichern von Dateien wird die Frage gestellt, ob die ganze Serie der Spektren mit den optimierten Parametern normiert werden soll. Beantwortet man diese Frage mit „y“, werden alle Spektren normiert, was einige Minuten dauern kann.

Nachfolgend wird der Ablauf an Hand eines Beispiels dokumentiert. Es wird ein Echellespektrum von bet Aur normiert. Man startet eine Konsole und geht in das Verzeichnis, welches das Spektrum enthält. Dann startet man ipython mit dem Pfad zu dem Skript:

1. Schritt: Programmstart

```
(base) lothar@Infinity:~/aa/Berthold/betAur/20200408$ ipython ~/Arbeitsordner/PythonProjekte/Codeschnipse
```

```
Path and name of the spectra (use wildcards) : _beta_aurig2200408start_20200408_761_full.fit
```

Worauf das Programm in der Konsole antwortet:

```
Number of spectra: 1
```

```

281 # ***** Change Smoothing Factor
282 print('\nCurrently the smoothing factor is "sm": ', sm)
283 print(
284     '\nWith the Smoothing factor we influence the "detail" of the spline: \nsm =
1 means that the spline goes through all points, \nsm = 0 means linear
regression through all points (straight line). \nSelect a number between 0. and
1.'
285 )
286
287 sm_Frage = input("\ndo you want to change the smoothing factor? Then enter y: ")
288
289 while sm_Frage == "y":
290     sm = float(input("Enter the new value for sm: "))
291     normfunction = csaps(points_wave[:,], points_flux[:,], wave[:,], smooth=sm)
292     fig = plt.figure(figsize=(10, 8))
293     plt.xlabel("Wavelength [Angström]")
294     plt.ylabel("normalized flux")
295     plt.title("Spectrum " + filelist[0])
296     plt.grid(True)
297     plt.plot(wave, flux)
298     plt.plot(points_wave, points_flux, "or", markersize=3)
299     plt.plot(wave, normfunction, "k")
300     plt.show()
301     sm_Frage = input("Do you want to change the smoothing factor again? Then
enter y: ")
302
303
304 # ***** Remove interpolation points below and above the spline
305 s1 = []
306 s2 = []
307 frage2 = []
308 j = 0
309 frage2.append(
310     input(
311         '\nif you want to delete interpolation points s1 % below and s2 % above
the spline, enter y: "
312     )
313 )
314 while frage2[j] == "y":
315     s1.append(float(input("Enter s1 in %: ")))
316     s2.append(float(input("Enter s2 in %: ")))
317     print("Please wait!")
318     points_flux = reduction(s1[j], s2[j], points_wave, points_flux, wave,
normfunction)
319
320     points_wave = points_wave[points_flux != 0]
321     points_flux = points_flux[points_flux != 0]
322
323     wave = cutting(wave, points_wave)
324
325     flux = flux[wave != 0]
326     wave = wave[wave != 0]
327     print("Current number of support points: ", len(points_wave))
328
329     normfunction = csaps(points_wave[:,], points_flux[:,], wave[:,], smooth=sm)
330
331     fig = plt.figure(figsize=(10, 8))
332     plt.xlabel("Wavelength [Angström]")
333     plt.ylabel("normalized flux")
334     plt.title("Spectrum " + filelist[0])
335     plt.grid(True)
336     plt.plot(wave, flux)
337     plt.plot(points_wave, points_flux, "or", markersize=3)
338     plt.plot(wave, normfunction, "k")
339     plt.show()
340
341     j += 1
342     frage2.append(
343         input(
344             '\nif you want to delete further interpolation points s1 % below
and s2 % above the spline, enter y: "
345         )
346     )
347
348
349 normspectrumflux = flux / normfunction
350
351 fig = plt.figure(figsize=(10, 8))
352 plt.xlabel("Wavelength [Angström]")
353 plt.ylabel("normalized flux")
354 plt.title("Spectrum " + filelist[0])
355 plt.grid(True)
356 plt.ylim(flux.min(), flux.max())
357 plt.plot(wave, flux)
358 plt.plot(points_wave, points_flux, "or", markersize=3)
359 plt.plot(wave, normfunction, "k")
360
361 frage3 = input("If the graphic is to be saved as pdf, then enter y: ")
362 if frage3 == "y":
363     filename = filelist[0].rsplit(".", 1)
364     filename = filename[0] + "_points.pdf"
365     plt.savefig(filename)
366
367
368 # Plot the normalized spectrum
369 fig = plt.figure(figsize=(10, 8))
370 plt.plot(wave, normspectrumflux)

```

Figure 24.4: Pythonskript zum automatischen Normieren von Spektren im fits-Format. Fortsetzung.

We first optimize the parameter "interval width" and select wavelength ranges in which no interpolation points are desired.

Current interval width "inter" is: 10

Current number of support points: 9356

Do you want to change the interval width "inter"? Then enter y:

Und das Diagramm in Abb. 24.6 wird als Fenster geöffnet (blau: das Spektrum, rote Punkte: die 9356 festgestellten lokalen Maxima (vorläufige Normierungsstützstellen)).

Die Stützstellen wurden in Intervallen von 10 Pixeln gesucht. Da viele Stützstellen auch in den Balmerlinien gefunden wurden erhöhen wir im nächsten Schritt die Intervallbreite auf 30, in dem wir die Frage nach einer Änderung der Intervallbreite bejahen und anschließend 30 eingeben. Das Programm antwortet mit

Do you want to change the interval width "inter"? Then enter y: y

Enter the new interval width: 30

Current number of support points: 6952

Do you want to change the interval width "inter" again? Then enter y:

Die Anzahl der gefundenen Stützstellen wird auf 6952 vermindert und das sich öffnende neue grafische Fenster zeigt das Spektrum mit den neuen Stützstellen (Abb. 24.7). Die Stützstellen in den Balmerlinien sind verschwunden. Die Frage nach einer weiteren Änderung der Intervallbreite verneinen wir, in dem wir nicht „y“ eingeben sondern eine andere Taste drücken (im Beispiel return), worauf das Programm danach fragt, ob wir in bestimmten Wellenlängenbereichen Stützstellen löschen möchten. Wir antworten mit „y“ (ja) und geben die gewünschten Wellenlängenbereiche ein, hier für die H α - und die H β -Linie, danach verneinen wir die Frage durch Drücken der return-Taste. Die Stützstellenzahl ist weiter auf 6738 vermindert.

Do you want to remove interpolation points in certain wavelength ranges? Then enter y: y

Initial wavelength of the range? :4800

End wavelength of the range? :4900

Do you want to remove further support points? Then enter y: y

Initial wavelength of the range? :6520

End wavelength of the range? :6580

Do you want to remove further support points? Then enter y:

```

370 plt.plot(wave, normspectrumflux)
371 plt.xlabel("Wavelength [Angström]")
372 plt.ylabel("normalized flux")
373 plt.title("Spectrum " + filelist[0])
374 plt.grid(True)
375 if frage3 == "y":
376     filename = filelist[0].rsplit(".", 1)
377     filename = filename[0] + "_normalized.pdf"
378     plt.savefig(filename)
379
380
381 # saving the normalized spectrum as fit
382 antwort = input("\nIf you want to save the normalized spectrum as fit, then
enter y: ")
383 if antwort == "y":
384     header["CRVAL1"] = wave[0]
385     header["NAXIS1"] = len(wave)
386     header["CRPIX1"] = 1
387     filename = filelist[0].rsplit(".", 1)
388     filename = filename[0] + "_normalized.fit"
389     fits.writeto(
390         filename, normspectrumflux, header, overwrite=True,
391         output_verify="ignore"
392     )
393 # saving the normfunction
394 antwort1 = input("If you want to save the normfunction as fit, then enter y: ")
395 if antwort1 == "y":
396     header["CRVAL1"] = wave[0]
397     header["NAXIS1"] = len(wave)
398     header["CRPIX1"] = 1
399     filename = filelist[0].rsplit(".", 1)
400     filename = filename[0] + "_normfunction.fit"
401     fits.writeto(filename, normfunction, header, overwrite=True)
402
403 # ***** Normalize entire time series *****
404
405
406 frage1 = input(
407     "\nIf you want to normalize the other spectra of the time series, enter y: "
408 )
409 print(
410     "If your time series contains many (> 10) spectra, the displayed graphs of
raw and normalized spectra may block a lot of memory."
411 )
412 frage4 = input(
413     "Do you want to calculate and to see all plots with the raw spectra,
normalization points and the normalization function? Then enter y: "
414 )

```

```

normalization points and the normalization function? Then enter y: "
414 )
415 frage5 = input(
416     "Do you want to calculate and to see all plots with the normalized spectra?
Then enter y: "
417 )
418 print(
419     "\nNow please wait until the program is finished. This may take some time.
With Ctrl C it can be aborted. The normalized spectra and pdf of the diagrams
can be found in the working folder."
420 )
421
422 if __name__ == "__main__":
423     if frage1 == "y":
424         with futures.ProcessPoolExecutor(max_workers=os.cpu_count() - 1) as e:
425             fs = {e.submit(pipe, n): n for n in np.arange(1, len(filelist))}
426             fertig = False
427             while not fertig:
428                 res = futures.wait(fs)
429                 for f in res.done():
430                     (
431                         k,
432                         normfunction,
433                         normspectrumflux,
434                         points_flux,
435                         points_wave,
436                         wave,
437                         flux,
438                         header,
439                     ) = f.result()
440                     del fs[f]
441                 # Saving the normalized spectrum as fit
442                 if antwort == "y":
443                     filename = filelist[k].rsplit(".", 1)
444                     filename = filename[0] + "_normalized.fit"
445                     header["CRVAL1"] = wave[0]
446                     header["NAXIS1"] = len(wave)
447                     header["CRPIX1"] = 1
448                     fits.writeto(
449                         filename,
450                         normspectrumflux,
451                         header,
452                         overwrite=True,
453                         output_verify="ignore",
454                     )
455                 # saving the normfunction
456                 if antwort1 == "y":
457                     header["CRVAL1"] = wave[0]
458                     header["NAXIS1"] = len(wave)

```

```

459     header["CRPIX1"] = 1
460     filename = filelist[k].rsplit(".", 1)
461     filename = filename[0] + "_normfunction.fit"
462     fits.writeto(filename, normfunction, header,
463                 overwrite=True)
464
465     # plotting
466     if frage4 == "y":
467         # plot
468         fig = plt.figure(figsize=(10, 8))
469         plt.ylim(flux.min(), flux.max())
470         plt.plot(wave, flux)
471         plt.xlabel("Wavelength [Angström]")
472         plt.ylabel("ADU")
473         plt.title("Spectrum " + filelist[k])
474         plt.grid(True)
475         plt.plot(points_wave, points_flux, "or", markersize=3)
476         plt.plot(wave, normfunction, "k")
477         if frage3 == "y":
478             filename = filelist[k].rsplit(".", 1)
479             filename = filename[0] + "_points.pdf"
480             plt.savefig(filename)
481             plt.show()
482
483         # Plot the normalized spectrum
484         fig = plt.figure(figsize=(10, 8))
485         plt.plot(wave, normspectrumflux)
486         plt.xlabel("Wavelength [Angström]")
487         plt.ylabel("normalized flux")
488         plt.title("Spectrum " + filelist[k])
489         plt.grid(True)
490         if frage3 == "y":
491             filename = filelist[k].rsplit(".", 1)
492             filename = filename[0] + "_normalized.pdf"
493             plt.savefig(filename)
494             plt.show()
495
496     fertig = len(res.not_done) == 0
497
498 x = input("Type e when you are finished viewing the graphics: ")
499 while x:
500     if x == "e":
501         print("END of program, all done")
502         x = False
503     else:
504         x = input(
505             "The input was not e.Type e when you are finished viewing the
graphics: "
506         )
507

```

Figure 24.5: Pythonskript zum automatischen Normieren von Spektren im fits-Format. Fortsetzung.

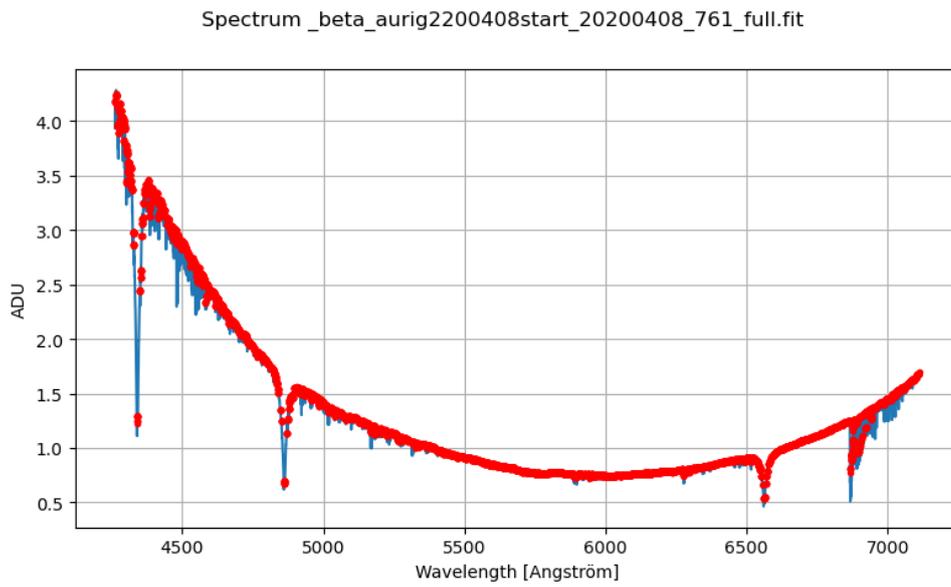


Figure 24.6: Grafik nach Start des Skripts.

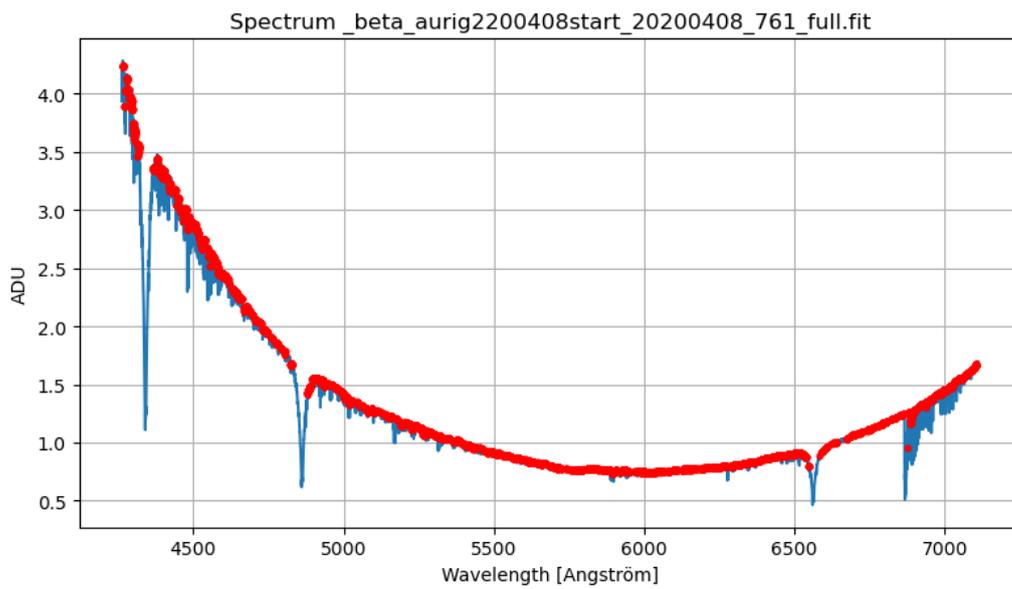


Figure 24.7: Grafik nach Änderung der Intervallbreite von 10 auf 30 Pixel.

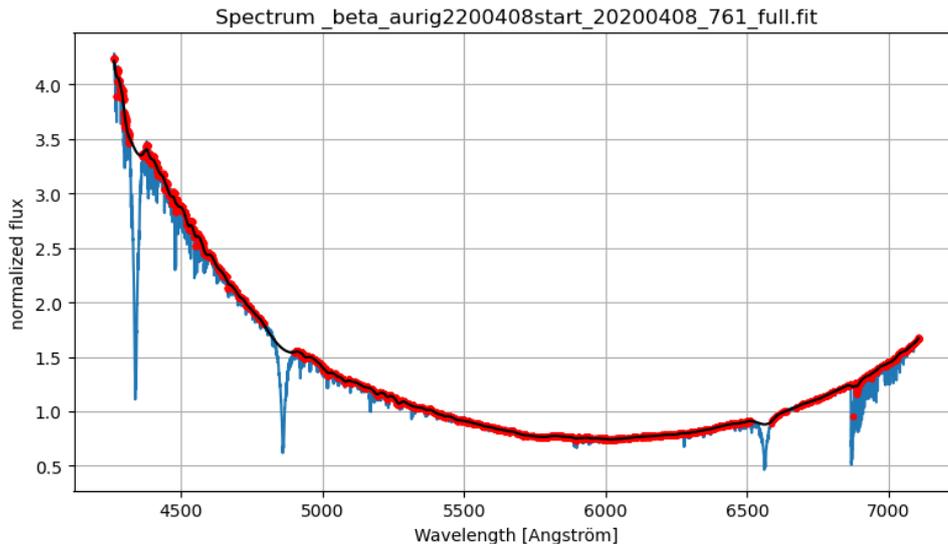


Figure 24.8: Grafik nach Ausschluß der Wellenlängenbereiche um die H α - und der H β -Linie. Der berechnete Spline ist als schwarze Kurve eingezeichnet.

Current number of support points: 6738

Daraufhin berechnet das Programm einen Spline als Vorschlag für eine Normierungsfunktion und zeigt diesen als schwarze Linie in dem sich öffnenden Grafikfenster (Abb. 24.8).

Der Spline ist zu „weich“ eingestellt, er senkt sich in die Balmerlinien ein. Deshalb bejahen wir die Frage nach einer Änderung des Smoothingfaktors sm mit „y“.

Currently the smoothing factor is "sm": 0.001

With the Smoothing factor we influence the "detail" of the spline: $sm = 1$ means that the spline goes through all points, $sm = 0$ means linear regression through all points (straight line).

Select a number between 0. and 1.

Do you want to change the smoothing factor? Then enter y: y

Enter the new value for sm: 0.0001

Do you want to change the smoothing factor again? Then enter y: y

Enter the new value for sm: 0.00001

Do you want to change the smoothing factor again? Then enter y: y

Enter the new value for sm: 0.000001

Nachdem wir mehrfach den Smoothingfaktor sm um eine Zehnerpotenz verringert haben, geht der Spline glatt über die Balmerlinien hinweg, folgt aber immer noch gut dem Kontinuumverlauf. Wir belassen es dabei.

Als nächstes wird die Frage gestellt, ob noch Stützstellen in einem Band um den Spline gelöscht werden sollen. Da immer noch Stützstellen in den terrestrischen Linien bei 6800 Å deutlich unter dem Spline zu finden sind, bejahen wir die Frage und geben die Löschgrenzen in % des Fluxwertes ein. Hier 0.5% und 5%. Das sich neu öffnende Grafik-Fenster zeigt nun den neuen Stützstellensatz an, die Punkte unter dem Spline im Bereich um 6800 Å sind verschwunden, die Anzahl der Stützstellen weiter vermindert auf jetzt 5451 (Abb. 24.10).

If you want to delete interpolation points $s1$ % below and $s2$ % above the spline, enter y: y

Enter $s1$ in %: 0.5

Enter $s2$ in %: 5

Please wait

Current number of support points: 5451

If you want to delete further interpolation points $s1$ % below and $s2$ % above the spline, enter y:

Wir sind jetzt am Ende der Optimierung der Parameter angekommen und beantworten noch einige Fragen bzgl. der Anzeige und Speicherung von Grafiken und Dateien und beenden das Programm mit der Eingabe von „e“, wenn wir mit dem betrachten und evtl. modifizieren der Grafiken fertig sind.

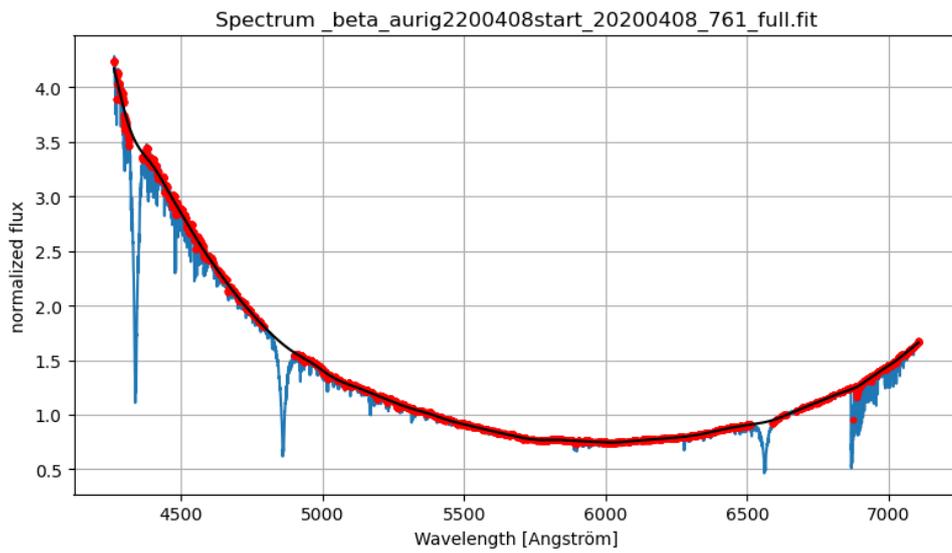


Figure 24.9: Spline nach Anpassung des Smoothingfaktors sm auf 0.000001.

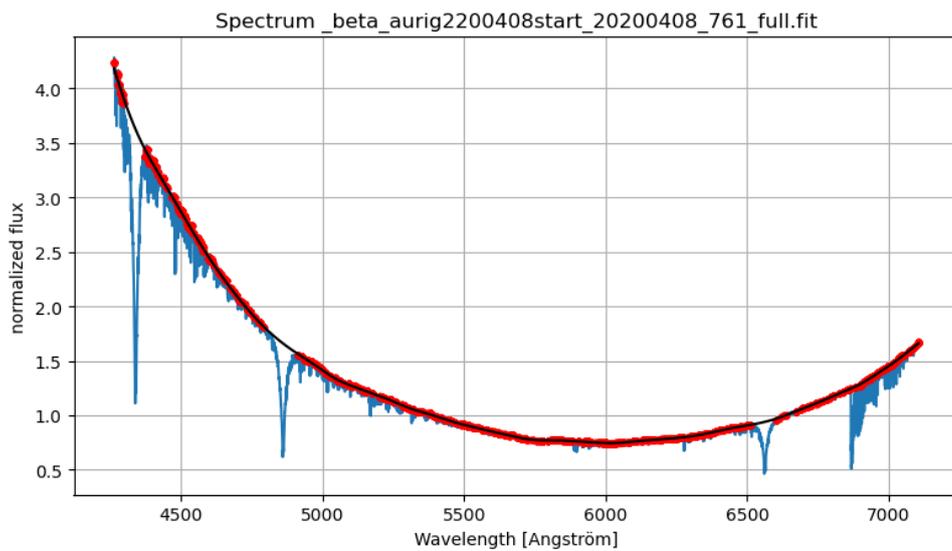


Figure 24.10: Grafik nach Löschen von Stützstellen außerhalb des Bands -0.5% bis $+5\%$ des Splines.

If the graphic is to be saved as pdf, then enter y:

If you want to save the normalized spectrum as fit, then enter y: y

If you want to save the normfunction as fit, then enter y: y

If you want to normalize the other spectra of the time series, enter y:

If your time series contains many (> 10) spectra, the displayed graphs of raw and normalized spectra may block a lot of memory.

Do you want to calculate and to see all plots with the raw spectra, normalization points and the normalization function? Then enter y:

Do you want to calculate and to see all plots with the normalized spectra? Then enter y:

Now please wait until the program is finished. This may take some time. With Ctrl C it can be aborted.

The normalized spectra and pdf of the diagrams can be found in the working folder.

Type e when you are finished viewing the graphics: e

END of program, all done

Die einzelnen Schritte wurden im Verlauf der Optimierung der Parameter für das erste Spektrum gespeichert. Wird die Frage nach der Normierung der anderen Spektren der Serie bejaht, werden alle Spektren individuell mit dem gespeicherten workflow normiert, was einige Minuten Wartezeit erfordern kann.

Part VII.

Bestimmung des Signal-Rausch-Verhältnisses

Das Signal-Rausch-Verhältnis (SNR = signal noise ratio) eines 1d-Spektrums im fits-Format kann mit dem Skript *SNR_fits_mehrereBereiche.py* interaktiv bestimmt werden. Nach der Eingabe des Pfads zum Spektrum öffnet sich ein interaktives Grafikfenster, in dem Wellenlängenbereiche, für die das SNR gemessen werden soll, angeklickt werden. Die SNR der Bereiche werden in einer ascii-Datei (Format tab) gespeichert, der Mittelwert wird in der Konsole ausgedruckt.

Part VIII.

Baryzentrische Korrektur (barycentric correction BC)

25. Berechnung der BC

Für die Berechnung der BC wird nachfolgend eine geringfügige Abwandlung einer Prozedur aus *PyAstronomy* vorgestellt. Das Skript BC.py verwendet aus *PyAstronomy* den Modul *pyasl*. Benötigt werden die Koordinaten des Objekts und des Beobachters (Observatory) sowie der Zeitpunkt der Beobachtung als JD.

Das Skript berücksichtigt zwei praktisch wichtige Fälle:

1. Man möchte die BC für *ein* Spektrum berechnen. Die Eingabedaten werden nacheinander abgefragt (Teil I).

2. Man hat eine Serie von 1d-Spektren eines Objekts, für die BC-Korrekturen berechnet werden sollen. Dann sind die Beobachter- und die Objektkoordinaten konstant, nur der Beobachtungszeitpunkt variiert. Dann ist es einfacher, Teil II zu verwenden und dort die Parameter im Skript einzutragen

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 SNR_fits_mehrereBereiche.py
5
6 View a wavelength calibrated 1d spectrum,
7 select the ranges in the wavelength scale by 2 left mouseclicks and put twice
8 the Escape-Button.
9 Result will be printed out.
10
11 Release 20211122
12
13 Author: Lothar Schanne
14 """
15
16 import numpy as np
17 from astropy.io import fits, ascii
18 import matplotlib.pyplot as plt
19
20 # Path and name of the fits-file
21 file = input("Path and name of the fits-file: ")
22
23 # Read header and data (flux)
24 flux, header = fits.getdata(file, header=True)
25
26 print("Minimum and Maximum in flux: ", flux.min(), " ", flux.max())
27
28 # Check for the necessary header entries
29 nax = header["NAXIS1"]
30 crval = header["CRVAL1"]
31 cdel = header["CDEL1"]
32 if "CRPIX1" not in header:
33     header["CRPIX1"] = 1
34
35 # Generation of a numpy array with the wavelengths of the spectrum
36 wave = np.ones(nax, dtype=float)
37 crval = crval + (1 - header["CRPIX1"]) * cdel
38 for i in range(nax):
39     wave[i] = crval + i * cdel
40 # The wave list contains the wavelengths of the pixels.
41 # In the list flux the corresponding intensities.
42
43 # Plot spectrum
44 plt.style.use("seaborn-white")
45 fig = plt.figure(1, figsize=(14, 10))
46 plt.plot(wave, flux)
47 plt.xlabel("Wavelength [Angstrom]")
48 plt.ylabel("ADU")
49 plt.title("Spectrum " + file)
50 plt.grid(True)
51
52
53 # Interactive setting of the base points begin and end for SNR estimation
54 # Press the ESC key twice to end the interactivity
55 plt.waitforbuttonpress()
56
57 snr = []
58 anfang = []
59 ende = []
60 eingabe = "y"
61 while eingabe == "y":
62     print("nach Markieren der beiden Wellenlängen im Diagramm per linken ")
63     print("Mausklick zweimal escape-Taste drücken: ")
64     pts = []
65     pts = np.asarray(plt.ginput(n=-1, timeout=-1))
66     if plt.waitforbuttonpress():
67         pass
68     plt.plot(pts[:, 0], pts[:, 1], "o", markersize=6)
69
70 a = pts[0, 0]
71 b = pts[1, 0]
72
73 aindex = int((a - crval) / cdel)
74 bindex = int((b - crval) / cdel)
75
76 newflux = flux[aindex:bindex]
77 newwave = wave[aindex:bindex]
78
79 # Calculation of SNR:
80 SNR = newflux.mean() / newflux.std()
81 snr.append(SNR)
82 anfang.append(a)
83 ende.append(b)
84 print("SNR zwischen %.2f und %.2f = %.1f" % (a, b, SNR))
85 print("\nNeuen Bereich eingeben? Dafür y eingeben.")
86 print("Zum Programmabbruch einen anderen Buchstaben eingeben")
87 eingabe = input()
88
89 # Abspeichern als ascii-Datei
90 ascii.write(
91     [anfang, ende, snr],
92     file + ".SNR.dat",
93     overwrite=True,
94     names=["anfang", "ende", "SNR"],
95     format="tab",
96 )
97
98 SNR = 0
99 for i in range(len(snr)):
100     SNR += snr[i]
101 SNR_mean = SNR / len(snr)
102 print("Mittleres SNR = ", SNR_mean)

```

Figure 24.11: Skript zur Bestimmung des SNR in mehreren Wellenlängenbereichen eines 1d-Spektrums.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Berechnung der Baryzentrischen Korrektur
5 abgeleitet aus einer Prozedur aus PyAstronomy
6 Entweder jedes Mal Eingabe aller nötigen Daten (Teil I bei auskommentierten
7 Teil II) oder Vorfütterung des Skripts in Teil II nach Auskommentierung von
8 Teil I, wenn die meisten Daten konstant bleiben (Ort der Beobachtung, Stern).
9
10 Author: Lothar Schanne
11 Stand 20180907
12 """
13
14 from PyAstronomy import pyasl
15
16 # Teil I
17 # Eingabe der Daten
18 # print("Eingabe der Koordinaten Observatory in folgender Form:")
19 # print(longitude = 209.5967661 in Grad)
20 # print(latitude = -24.62586583 in Grad)
21 # print(altitude = 2635.43 in Meter)
22 # longitude = float(input("Eingabe der Koordinaten Observatory, longitude:
23 ))
24 # latitude = float(input("Eingabe der Koordinaten Observatory, latitude:
25 ))
26 # altitude = float(input("Eingabe der Koordinaten Observatory, altitude:
27 ))
28
29 # print("Eingabe der Koordinaten Stern in folgender Form:")
30 # print(coordinates of HD 22345 (J2000))
31 # print(ra2000 = 496.20313477)
32 # print(dec2000 = -12.87498346)
33 # print(ra2000 = float(input("ra2000 Stern: "))
34 # print(dec2000 = float(input("dec2000 Stern: "))
35 #
36 # print("Zeitpunkt der Beobachtung als JD in der Form 2458528.2335")
37 # jd = float(input("zeitpunkt: "))
38
39 # Teil II
40 # Konkretes Fall, bei öfterer Anwendung gleicher Daten zu bevorzugen
41 # Coordinates of European Southern Observatory
42 # (Coordinates of UT)
43 # longitude = 209.5967661
44 # latitude = -24.62586583
45 # altitude = 2635.43
46
47 # Coordinates of HD 22345 (J2000)
48 # ra2000 = 496.20313477
49 # dec2000 = -12.87498346
50
51 # (Mid-Time of observation
52 # jd = 2458528.2335
53
54 # Calculate barycentric correction (debug=True show
55 # various intermediate results)
56
57 corr, hjd = pyasl.helcorr(longitude, latitude, altitude, \
58 ra2000, dec2000, jd, debug=True)
59
60 print("Barycentric correction [km/s]: ", corr)
61 print("heliocentric Julian day: ", hjd)

```

Figure 25.1: Berechnung der Baryzentrischen Korrektur mit dem Skript BC.py.

(Teil I vorher auskommentieren, wie in Abb. 25.1 gezeigt).

Neben der BC und heliozentrischen Korrektur werden vorher viele (interessante) Zwischenwerte ausgegeben. Das lässt sich in Zeile 58 des Parameters *debug* auf *False* ändern.

26. BC-Korrektur einer Serie von 1d-Spektren im fits-Format durch Dopplerverschiebung

Besitzt man eine Zeit-Serie von 1d-fits-Dateien, welche vom gleichen Standort am gleichen Objekt gemessen wurden, möchte man vielleicht die Spektren baryzentrisch korrigieren (um beispielsweise anschließend Radialgeschwindigkeitsmessungen daran vorzunehmen). Dann ist ein Programm praktisch, dass dies für alle Spektren auf einmal erledigt. Das Skript *BC_Dopplershift_1dSpectrum_Serie.py* ist dafür geeignet. Es ist in Abb. 26.1 gelistet.

Es benötigt neben den geografischen Koordinaten für den Beobachtungsstandort und die Himmelskoordinaten des Objekts, die im Skript anzupassen sind. Das Programm speichert die baryzentrisch korrigierten Spektren in drei Formaten ab: fit, dat und csv. Die Dateien werden neu benannt, mit der Erweiterung *_bc* am Ende ihres ursprünglichen Bezeichners. Im fit ist der Header um die Angabe der BC erweitert.

Part IX.

Bestimmung von Radialgeschwindigkeiten

Die Bestimmung von Radialgeschwindigkeiten (RV's) von Himmelsobjekten ist eine häufige Messaufgabe. Dies geschieht durch Messung der Wellenlänge einzelner Linien und Vergleich mit der Laborwellenlänge oder durch Kreuzkorrelation von Spektrumausschnitten mit einem Template (theoretisches Spektrum, das keine Dopplerverschiebung ($RV = 0$) besitzt). Die verwendeten Spektren sind üblicherweise auf das Kontinuum normiert. Die so erhaltene RV muß noch heliozentrisch korrigiert werden, damit die Rotation und Eigengeschwindigkeit der Erde um die Sonne eliminiert werden.

Die meisten Absorptionslinien sind symmetrisch. Ihre charakteristische Wellenlänge, aus der dann die RV berechnet wird, wird dann durch das Minimum der Linie definiert. Das Minimum einer Linie kann

- interaktiv bestimmt werden, in dem man z.B. in einer grafischen Darstellung der Absorptionslinie das Minimum mit der Maus anklickt.
- berechnet werden, in dem man mittels einer mathematischen Methode das Linienprofil modelliert (fitting) und im Modell das Minimum berechnet.

Für beide Verfahren wurden Pythonskripte entwickelt, die nachfolgend beschrieben werden.

27. Interaktive Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im ascii-Format (tab-separiert)

Die Spektrenserie liegt im ascii-Format vor, zwei Spalten (WAVE und FLUX), tab-separiert. Das Programm fragt zuerst nach dem Pfad und den Namen der Spektren (Zeile 25, Abb. 27.1).

Nach dem Ausdruck einer umfangreichen Liste wählbarer Linien³¹ in der Konsole wird man gefragt (Zeile 168, Abb. 27.2), welche Linie man wählen möchte und gibt die gewünschte Linie genau so ein, wie sie in der Liste aufgeführt wird. In Zeile 190 des Skripts wird ein Suchintervall in der Einheit Angström definiert, innerhalb dem die Linie um die Laborwellenlänge gesucht wird. Sie kann im Skript

³¹Die Liste kann natürlich im Skript durch weitere Linien ergänzt werden.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 BC_Dopplershift_1dSpectrum_Serie.py
5
6 Das Skript korrigiert eine Serie von 1d-Spektrum im fits-Format eines Objekts
7 an einem Standort gemessen um die baryzentrische Korrektur.
8 Schreibt 3 files (fit, csv und dat) der BC-korrigierten Spektren ins
9 Arbeitsverzeichnis. In den erzeugten fit ist die jeweilige BC vermerkt.
10 Der Beobachtungszeitpunkt muss in den id-fit im Header als Parameter namens
11 'JD-OBS' als JD in Format float enthalten sein. Falls der Parameter fehlt wird
12 danach gefragt und ist per Tastatur einzugeben. Er wird dann auch in die
13 BC-korrigierten .fit im Header eingetragen.
14
15 @author: Lothar Schanne
16 Stand 20180908
17 """
18
19 from PyAstronomy import pyasl
20 import numpy as np
21 from astropy.io import fits
22 from astropy.io import ascii
23 import glob
24
25
26 # Koordinaten des Observatory in folgender Form:
27 # longitude = 289.5967661 in Grad
28 # latitude = -24.62586583 in Grad
29 # altitude = 2635.43 in Meter
30
31 # Hier als Beispiel Koordinaten von Berthold
32 # BITTE AUF EIGENE KOORDINATEN ÄNDERN
33 longitude = 7.4775
34 latitude = 49.475277777778
35 altitude = 200
36
37 # Eingabe der Koordinaten des Sterns in folgender Form:
38 # ra2000 = 030.20913477
39 # dec2000 = -12.87490346
40
41 # Hier als Beispiel Koordinaten von del Cep, BITTE FÜR OBJEKT ÄNDERN
42 ra2000 = 337.292771
43 dec2000 = +58.415198
44
45 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
46 files = input('Pfad und Name der Spektren (nutze wildcards): ')
47 filelist = glob.glob(files)
48
49 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
50 zeitliche Ordnung
51 filelist.sort()
52
53 # Ausdruck der Liste
54 print('\nSpektrenliste: \n')
55 print(filelist)

```

```

53 # Ausdruck der Liste
54 print('\nSpektrenliste: \n')
55 print('Anzahl der Spektren: ', len(filelist), '\n')
56
57 # Abarbeiten der filelist, Einlesen von flux und header:
58 for i in range(len(filelist)):
59     flux, header = fits.getdata(filelist[i], header=True)
60     # Prüfung auf die nötigen header-Einträge:
61     print(filelist[i], ' ')
62     if 'NAXIS1' in header:
63         print('Dimension, NAXIS1: ', header['NAXIS1'])
64     else:
65         print('Das ist kein 1d-Spektrum !')
66     if 'NAXIS1' in header:
67         nax = header['NAXIS1']
68         print('Anzahl der Werte (Abszisse), NAXIS1: ', nax)
69     else:
70         print('NAXIS1 fehlt im header !')
71     if 'CRVAL1' in header:
72         crval = header['CRVAL1']
73         print('Anfangs-Wellenlänge, CRVAL1: ', crval)
74     else:
75         print('CRVAL1 fehlt im header !')
76     if 'CRPIX1' in header:
77         crpix = header['CRPIX1']
78         print('Referenzpixel, CRPIX1: ', crpix)
79     else:
80         print('CRPIX1 fehlt im header !')
81         crpix = 1
82     if 'CDELTA1' in header:
83         cdel = header['CDELTA1']
84         print('Schrittweite der Wellenlänge, CDELTA1: ', cdel)
85     else:
86         print('CDELTA1 fehlt im header !')
87     if 'JD-OBS' in header:
88         jd = float(header['JD-OBS'])
89         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
90     if 'JD-OBS' in header:
91         jd = float(header['JD-OBS'])
92         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
93     if 'JD' in header:
94         jd = float(header['JD'])
95         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
96
97 # Calculate barycentric correction (debug=True show
98 # various intermediate results)
99 corr, hjd = pyasl.helcorr(longitude, latitude, altitude,
100                          ra2000, dec2000, jd, debug=False)
101
102 print("Barycentric correction [km/s]: ", corr)
103 print("Heliocentric Julian day: ", hjd)
104
105 # Erzeugen von numpy-Arrays mit den Wellenlängen und Fluxes des Spektrums
106 wave = np.ones(nax, dtype=float)

```

```

77     crpix = header['CRPIX1']
78     print('Referenzpixel, CRPIX1: ', crpix)
79     else:
80         print('CRPIX1 fehlt im header !')
81         crpix = 1
82     if 'CDELTA1' in header:
83         cdel = header['CDELTA1']
84         print('Schrittweite der Wellenlänge, CDELTA1: ', cdel)
85     else:
86         print('CDELTA1 fehlt im header !')
87     if 'JD-OBS' in header:
88         jd = float(header['JD-OBS'])
89         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
90     if 'JD-OBS' in header:
91         jd = float(header['JD-OBS'])
92         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
93     if 'JD' in header:
94         jd = float(header['JD'])
95         print('Beobachtungszeitpunkt, JD: ', jd, '\n')
96
97 # Calculate barycentric correction (debug=True show
98 # various intermediate results)
99 corr, hjd = pyasl.helcorr(longitude, latitude, altitude,
100                          ra2000, dec2000, jd, debug=False)
101
102 print("Barycentric correction [km/s]: ", corr)
103 print("Heliocentric Julian day: ", hjd)
104
105 # Erzeugen von numpy-Arrays mit den Wellenlängen und Fluxes des Spektrums
106 wave = np.ones(nax, dtype=float)
107 for k in range(nax):
108     wave[k] = crval + (k - crpix + 1) * cdel
109
110 # Shift that spectrum redward by corr using
111 # "firstlast" as edge handling method.
112 flux_bc, wave_bc = pyasl.dopplershift(wave, flux, corr,
113                                     edgeHandling='firstlast')
114
115 # Schreiben des BC-korrigierten Spektrums in ascii-files
116 # ascii.write([wave_bc, flux_bc], filelist[i].rstrip('.fit')+'.bc.csv',
117 #            overwrite=True, names=['WAVE', 'FLUX'], format='csv')
118 # ascii.write([wave_bc, flux_bc], filelist[i].rstrip('.fit')+'.bc.dat',
119 #            overwrite=True, names=['WAVE', 'FLUX'], format='tab')
120 # Schreiben des BC-korrigierten Spektrums in fits-file
121 header['CRVAL1'] = wave_bc[0]
122 header['CRPIX1'] = 1
123 header['NAXIS1'] = len(wave_bc)
124 newfile = filelist[i].rstrip('.fit')+'.bc.fit'
125 header['BARYCOR'] = (corr, 'km/s, corrected')
126 header['JD-OBS'] = (jd, 'observation mid-time in JD')
127 fits.writeto(newfile, flux_bc, header, overwrite=True, output_verify='ignore')
128
129 print('Neue Anfangswellenlänge CRVAL1: ', header['CRVAL1'], '\n\n')
130

```

Figure 26.1: Programmlisting von *BC_Dopplershift_1dSpectrum_Serie.py*.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Liest Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im
5 tab-Format, 2 Spalten WAVE und FLUX). Festlegung des Linienminimums per
6 Interaktion und Bestimmung der Radialgeschwindigkeit aus dem Minimum.
7 Plottet alle Spektren zum markieren von bis zu 2 Linien-Minima und gibt
8 ermittelte Daten (RV und Apex) als ascii-Dateien (tab-separiert, als .dat) aus.
9
10 Created on 20211230
11
12 @author: lothar schanne
13 """
14
15 import numpy as np
16 from astropy.io import ascii
17 import glob
18 from PyAstronomy import pyasl
19 import matplotlib.pyplot as plt
20
21 # plt.style.use('seaborn-whitegrid')
22
23
24 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
25 files = input("Pfad und Name der Spektren (nutze wildcards): ")
26 filelist = glob.glob(files)
27
28 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
29 # zeitliche Ordnung
30 filelist.sort()
31
32 # Ausdruck der Liste
33 print("\nspektrenliste: \n")
34 print("Anzahl der Spektren: ", len(filelist), "\n")
35
36
37 # Liste der wählbaren Linien:
38 Linien = {
39     "CIV 7226": 7226.2,
40     "CIII 7037": 7037.25,
41     "HeI 6678": 6678.149,
42     "FeI 6678": 6677.9865,
43     "FeI 6634": 6633.7492,
44     "FeI 6609": 6609.1098,
45     "H alpha": 6562.817,
46     "FeI 6546": 6546.24,
47     "FeI 6516": 6516.0783,
48     "FeI 6463": 6462.725,
49     "FeII 6456": 6456.3805,
50     "FeI 6417": 6416.9386,
51     "FeI 6412": 6411.6592,
52     "FeI 6408": 6408.0272,
53     "FeI 6400": 6400.0008,
54     "FeI 6394": 6393.6009,
55     "FeI 6376": 6376.36,
56

```

```

54 "FeI 6394": 6393.6009,
55 "SiII 6371": 6371.36,
56 "SiII 6347": 6347.10,
57 "FeI 6265": 6265.1335,
58 "FeI 6256": 6256.3011,
59 "FeI 6255": 6254.581,
60 "FeI 6253": 6252.555,
61 "FeII 6248": 6247.559,
62 "FeI 6233": 6232.6408,
63 "FeI 6231": 6230.7226,
64 "FeI 6213": 6213.299,
65 "FeI 6200": 6200.3125,
66 "FeI 6192": 6191.556,
67 "FeI 6180": 6180.2038,
68 "NiI 6177": 6176.81,
69 "NiI 6175": 6175.367,
70 "FeI 6173": 6173.3352,
71 "FeII 6170": 6169.816,
72 "FeI 6170": 6169.597,
73 "FeI 6164": 6163.5441,
74 "CaI 6162": 6162.17,
75 "FeII 6149": 6149.231,
76 "FeII 6148": 6147.734,
77 "HgII 6142": 6141.773,
78 "FeI 6142": 6141.7316,
79 "FeI 6137": 6137.286,
80 "CaI 6122": 6122.22,
81 "NiI 6100": 6100.12,
82 "CaI 6103": 6102.72,
83 "FeI 6065": 6065.482,
84 "FeI 6056": 6056.0043,
85 "FeI 6027": 6027.0505,
86 "FeI 6024": 6024.0576,
87 "FeI 6020": 6020.1688,
88 "CuII 6013": 6013.411,
89 "FeIII 5920": 5920.0,
90 "CII 5920": 5919.6,
91 "FeIII 5919": 5918.968,
92 "Na D1": 5895.924,
93 "Na D2": 5895.951,
94 "HeI 5876": 5875.989,
95 "FeI 5860": 5859.608,
96 "FeI 5862": 5862.357,
97 "HeI 5861": 5861.35,
98 "CIV 5812": 5812.140,
99 "CIV 5802": 5801.510,
100 "CIII 5696": 5696.008,
101 "OIII 5592": 5592.376,
102 "OIII 5593": 5592.37,
103 "FeI 5576": 5576.0884,
104 "FeI 5573": 5572.842,
105 "FeI 5570": 5569.6127,
106 "FeI 5567": 5567.3907,
107 "FeI 5566": 5566.7036,

```

Figure 27.1: Skript zur interaktiven Bestimmung der RV an einer Linie.

```

107 "FeI 5566": 5565.9036,
108 "FeI 5560": 5560.2112,
109 "FeI 5558": 5557.9818,
110 "FeI 5555": 5554.8947,
111 "FeI 5526": 5525.5439,
112 "FeI 5498": 5497.5157,
113 "TiII 5491": 5490.7,
114 "FeI 5447": 5446.8743,
115 "FeI 5430": 5429.6964,
116 "TiII 5419": 5418.0,
117 "FeI 5415": 5415.1909,
118 "FeII 5411": 5411.970,
119 "HeII 5411": 5411.524,
120 "FeI 5406": 5405.7749,
121 "TiI 5404": 5404.11,
122 "FeI 5383": 5383.3688,
123 "TiII 5381": 5381.03,
124 "FeI 5367": 5367.466,
125 "FeII 5363": 5362.9098,
126 "FeI 5307": 5307.36,
127 "FeI 5302": 5302.299,
128 "TiII 5262": 5262.14,
129 "FeI 5233": 5232.94,
130 "MgI 5183": 5183.0042,
131 "MgI 5172": 5172.6843,
132 "MgI 5167": 5167.3216,
133 "FeI 5162": 5162.2725,
134 "FeII 5154": 5154.207,
135 "FeI 5097": 5096.9977,
136 "FeI 5075": 5074.748,
137 "TiII 5072": 5072.25,
138 "FeI 5065": 5065.0181,
139 "VI 5062": 5061.70,
140 "FeI 5018": 5018.4354,
141 "HeI 5016": 5015.6783,
142 "FeI 5007": 5007.275,
143 "FeI 5002": 5001.8633,
144 "HeI 4922": 4921.929,
145 "H beta": 4861.332,
146 "HeI 4713": 4713.143,
147 "HgII 4687": 4686.563,
148 "HeI 4686": 4685.682,
149 "CIII 4650": 4650.10,
150 "CIII 4647": 4647.400,
151 "MgI 4571": 4571.0956,
152 "FeII 4542": 4541.905,
153 "HeI 4542": 4541.59,
154 "He 4452": 4451.59,
155 "He 4471": 4471.686,
156 "HeI 4380": 4387.928,
157 "CIII 4380": 4380.24,
158 "H gamma": 4340.468,
159 "NiII 4200": 4200.020,
160 "HI 4102": 4101.737,

```

```

160 "HI 4102": 4101.737,
161 "SiIV 4089": 4088.863,
162 "HeI 4026": 4026.191,
163
164 print("Linienauswahl: ", Linien.keys())
165
166
167 # Eingabe der zu messenden Linie:
168 Linie = input("Geben Sie den Namen der zu messenden Linie ein: ")
169 wellenlaenge = Linien[Linie]
170
171 # Abarbeiten der filelist, Einlesen von flux und Wellenlängen, Auswahl des Flux um die
172 # Linie:
173
174 # Definition von Variablen
175 RV1 = np.zeros(len(filelist))
176 RV2 = np.zeros(len(filelist))
177 apex1 = np.zeros(len(filelist))
178 apex2 = np.zeros(len(filelist))
179 minwave1 = np.zeros(len(filelist))
180 minwave2 = np.zeros(len(filelist))
181 miniflux1 = np.zeros(len(filelist))
182 miniflux2 = np.zeros(len(filelist))
183
184
185 for i in range(len(filelist)):
186     ts = ascii.read(filelist[i], format="tab")
187
188     # -----
189     # Breite des Suchintervalls, Breite in Angström, anpassen.
190     suchintervall = 4
191     # -----
192
193     intervall = np.extract(abs(ts.columns[0] - wellenlaenge) < suchintervall, ts)
194     a = np.zeros(len(intervall))
195     b = np.zeros(len(intervall))
196     for j in range(len(intervall)):
197         a[j], b[j] = intervall[j]
198
199     fig = plt.figure()
200     plt.plot(a, b, , linewidth=5)
201     plt.title(filelist[i].rstrip(".dat") + " _ " + Linie)
202
203     print("\nspektrum ", filelist[i])
204     print("Klicke das Minimum der ersten Linie an: ")
205     pts = np.asarray(plt.ginput(n=1, timeout=1))
206     RV1[i] = (pts[0, 0] - wellenlaenge) / wellenlaenge * 299792
207     apex[i] = pts[0, 1]
208     print("Erste Linie, RV =", RV1[i], "Apex =", apex[i])
209
210     frage = input(
211         "Möchten Sie das Minimum einer zweiten Linie anklicken? Dann y eingeben: "
212     )
213     if frage == 'y':

```

Figure 27.2: Skript zur interaktiven Bestimmung der RV an einer Linie (Fortsetzung).

```

177 minivave1 = np.zeros(len(filelist))
180 minivave2 = np.zeros(len(filelist))
181 miniflux1 = np.zeros(len(filelist))
182 miniflux2 = np.zeros(len(filelist))
183
184
185 for i in range(len(filelist)):
186     ts = ascii.read(filelist[i], format="tab")
187
188     # *****
189     # Breite des Suchintervalls, Breite in Angström, anpassen.
190     suchintervall = 4
191     # *****
192
193     intervall = np.extract(abs(ts.columns[0] - wellenlaenge) < suchintervall, ts)
194     a = np.zeros(len(intervall))
195     b = np.zeros(len(intervall))
196     for j in range(len(intervall)):
197         a[j], b[j] = intervall[j]
198
199     fig = plt.figure(i)
200     plt.plot(a, b, ".", linewidth=0.5)
201     plt.title(filelist[i].rstrip(".dat") + "." + linie)
202
203     print("\nSpektrum ", filelist[i])
204     print("Klicke das Minimum der ersten Linie an: ")
205     pts = np.asarray(plt.ginput(n=1, timeout=-1))
206     RV1[i] = (pts[0, 0] - wellenlaenge) / wellenlaenge * 299792
207     apex1[i] = pts[0, 1]
208     print("Erste Linie, RV =", RV1[i], "Apex =", apex1[i])
209
210     frage = input(
211         "Möchten Sie das Minimum einer zweiten Linie anklicken? Dann y eingeben: "
212     )
213
214     if frage == "y":
215         print("Klicke die zweite Linie an")
216         pts = np.asarray(plt.ginput(n=1, timeout=-1))
217         RV2[i] = (pts[0, 0] - wellenlaenge) / wellenlaenge * 299792
218         apex2[i] = pts[0, 1]
219         print("Zweite Linie, RV =", RV2[i], "Apex =", apex2[i])
220     else:
221         RV2[i] = np.NaN
222         apex2[i] = np.NaN
223
224     # plt.savefig(filelist[i].rstrip(".dat") + "." + linie + ".png")
225
226 # Abspeichern als ascii-Datei
227 ascii.write(
228     [filelist, RV1, apex1, RV2, apex2],
229     linie + "_RV_interaktiv" + ".dat",
230     overwrite=True,
231     names=["Spektrum", "RV1", "Apex1", "RV2", "Apex2"],
232     format="tab",
233 )

```

Figure 27.3: Skript zur interaktiven Bestimmung der RV an einer Linie (Fortsetzung).

angepasst werden. Man wird aufgefordert, das Minimum der gewählten Linie mit der Maus anzuklicken, danach werden die RV und der Apex des Minimums in der Konsole ausgedruckt. Daraufhin wird man gefragt, ob man das Minimum einer zweiten Linie anklicken möchte. Dies wird benötigt, wenn man die Spektren eines SB2-Sterns bearbeitet, also eines Doppelsterns, in dem die Spektren beider Sterne dopplerverschoben in den Spektren zu sehen sind. Falls man das wünscht, gibt man „y“ ein, ansonsten drückt man eine andere Taste, z.B. „return“. Wenn man Zeile 223 auskommentiert, wird das Grafikenster mit dem Spektrumausschnitt und den angeklickten Minima als png-Bild gespeichert. Nach der Bearbeitung aller Spektren der Serie werden die RV's und Apexe in einer ascii-Datei (.dat) gespeichert und das Programm ist beendet.

Eine Variante des Skripts namens *RV_MessungAnLinie_Zeitreihe_dat_perInteraktion_regression_2Linien.py*, in dem mit dem Mausclick auf ein Linienminimum eine Regression 4ten Grades im Bereich des Minimums angestoßen wird, um das Minimum genauer zu bestimmen, ist in Abb. 27.4 zu sehen. Das Skript unterscheidet sich erst ab Zeile 174 vom vorherigen. Die beiden Linien des SB2-Sterns müssen separiert sein, also 2 getrennte Minima zu erkennen sein, damit die Minimumerkennung per Regression funktioniert. Die Grafikenster mit den eingeblendeten Regressionskurven und dem bestimmten Minimum werden zur Kontrolle routinemäßig als png-Bild abgespeichert und die Ergebnisse werden wiederum als ascii-Datei abgespeichert.

In Abb. 27.5 ist das grafische Ergebnis des Skripts für die FeI 6463 Linie eines SB2-Systems (6 Tri A) gezeigt. Die beiden Linien sind im Minimumbereich durch eine Regressionskurve (rot und grün) modelliert und die Minima als schwarzer Punkt markiert.

28. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format

Das Minimum einer Linie kann durch Fitting des Linienprofils mittels eines Modells näherungsweise bestimmt werden. An Modellen stehen in Pythonbibliotheken zur Verfügung:

- Regression
- Radial basis functions

```

173 # Definition von Variablen
174 RV1 = np.zeros(len(filelist))
175 RV2 = np.zeros(len(filelist))
176 miniwave1 = np.zeros(len(filelist))
177 miniwave2 = np.zeros(len(filelist))
178 miniwave2 = np.zeros(len(filelist))
179 miniflux1 = np.zeros(len(filelist))
180 miniflux2 = np.zeros(len(filelist))
181
182
183 for i in range(len(filelist)):
184     ts = ascii.read(filelist[i], format="tab")
185
186     # *****
187     # Breite des Suchintervalls, Breite in Angström, anpassen.
188     suchintervall = 3
189     # *****
190
191     intervall = np.extract(abs(ts.columns[0] - wellenlaenge) < suchintervall, ts)
192     a = np.zeros(len(intervall))
193     b = np.zeros(len(intervall))
194     for j in range(len(intervall)):
195         a[j], b[j] = intervall[j]
196
197     fig = plt.figure(i)
198     plt.plot(a, b, "-", linewidth=0.5)
199     plt.title(filelist[i].rstrip(".dat") + "_" + linie)
200
201     print("\nSpektrum ", filelist[i])
202     print("Klicke das Minimum der ersten Linie an:")
203     pts = np.asarray(plt.ginput(n=1, timeout=-1))
204
205     # Das folgende Intervall [Angström] anpassen, bei verrauschten Linien breit,
206     # bei nicht verrauschten, schmalen Linien klein.
207     linienminimum_wave1 = a[abs(a - pts[0, 0]) <= 0.5]
208     linienminimum_flux1 = b[abs(b - pts[0, 0]) <= 0.5]
209
210     # Regression, Grad anpassen (2 oder 4 oder 6)
211     grad = 4
212     model = np.poly1d(np.polyfit(linienminimum_wave1, linienminimum_flux1, grad))
213
214     polyline = np.linspace(linienminimum_wave1[0], linienminimum_wave1[-1], 100)
215     modflux = model(polyline)
216
217     # Linienminimum berechnen
218     miniflux1[i] = modflux.min()
219     miniwave1[i] = polyline[modflux.argmax()]
220
221     # Plotten
222     plt.plot(polyline, modflux, "--")
223     plt.plot(miniwave1[i], miniflux1[i], "o", color="black")
224     plt.show()
225     plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")

```

```

219 miniwave1[i] = polyline[modflux.argmax()]
220
221 # Plotten
222 plt.plot(polyline, modflux, "--")
223 plt.plot(miniwave1[i], miniflux1[i], "o", color="black")
224 plt.show()
225 plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")
226
227 RV1[i] = (miniwave1[i] - wellenlaenge) / wellenlaenge * 299792
228
229 frage = input(
230     "Möchten Sie das Minimum einer zweiten Linie anklicken? Dann y eingeben: "
231 )
232
233 if frage == "y":
234     print("Klicke die zweite Linie an")
235     pts = np.asarray(plt.ginput(n=1, timeout=-1))
236
237     # Das folgende Intervall [Angström] anpassen, bei verrauschten Linien breit,
238     # bei nicht verrauschten, schmalen Linien klein.
239     linienminimum_wave2 = a[abs(a - pts[0, 0]) <= 0.3]
240     linienminimum_flux2 = b[abs(b - pts[0, 0]) <= 0.3]
241
242     # Regression
243     model = np.poly1d(np.polyfit(linienminimum_wave2, linienminimum_flux2, grad))
244
245     polyline = np.linspace(linienminimum_wave2[0], linienminimum_wave2[-1], 100)
246     modflux = model(polyline)
247
248     # Linienminimum berechnen
249     miniflux2[i] = modflux.min()
250     miniwave2[i] = polyline[modflux.argmax()]
251
252     # Plotten
253     plt.plot(polyline, modflux, "--")
254     plt.plot(miniwave2[i], miniflux2[i], "o", color="black")
255
256     RV2[i] = (miniwave2[i] - wellenlaenge) / wellenlaenge * 299792
257 else:
258     RV2[i] = np.NaN
259
260 plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")
261 plt.close(i - 1)
262
263 # Abspeichern als ascii-Datei
264 ascii.write(
265     [filelist, RV1, RV2],
266     linie + "_RV_interaktiv2lines" + ".dat",
267     overwrite=True,
268     names=["Spektrum", "RV1", "RV2"],
269     format="tab",
270 )

```

Figure 27.4: Skript zur interaktiven Bestimmung der RV an einer Linie, Variante mit Regression 4ten Grades zur Bestimmung des Minimums.

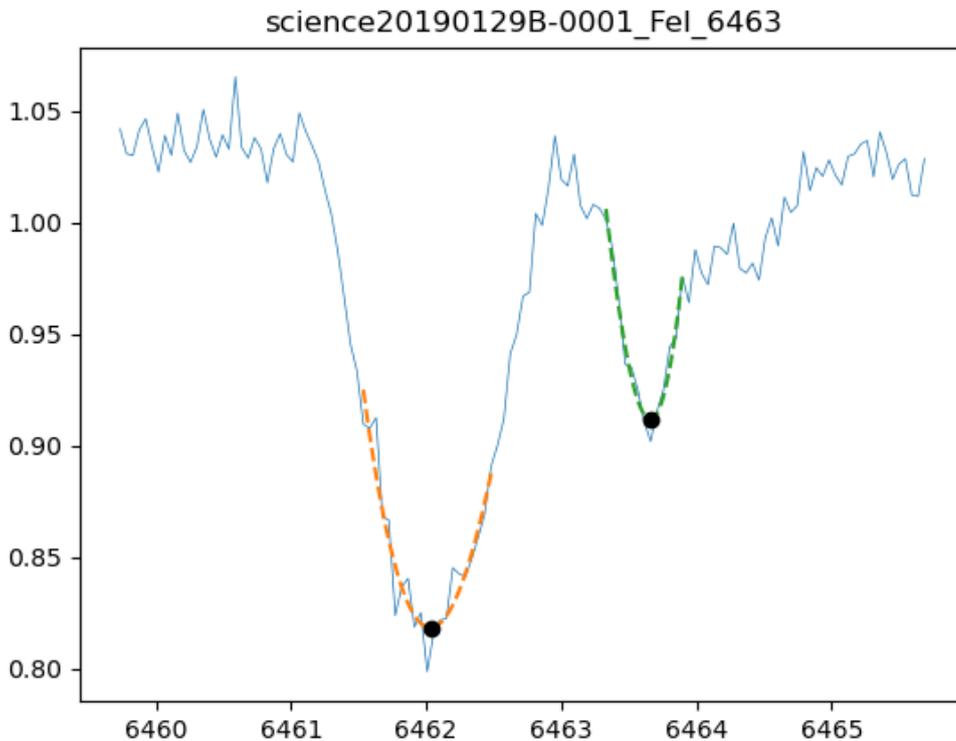


Figure 27.5: Grafische Darstellung der Minimumermittlung per Regressionskurve für die doppelte FeI 6463-Linie in einem Doppelsternspektrum (6 Tri A).

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Liest einen Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im
5 fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden)
6 Koordinaten des Beobachters und Objekts die heliozentrische Korrektur,
7 fitted die gewählte Linie im Minimumbereich per Regression
8 und bestimmt aus dem heliozentrisch korrigierten Minimum die
9 heliozentrisch korrigierte Radialgeschwindigkeit RV.
10 Plottet alle fittings und gibt ermittelte Daten als ascii-Dateien
11 (tab-separiert, als .dat) aus.
12
13 Created on Mon Mar  8 13:43:44 2021
14
15 @author: lothar
16 """
17
18 import numpy as np
19 from astropy.io import fits, ascii
20 import glob
21 import matplotlib.pyplot as plt
22 from PyAstronomy import pyasl
23
24 # plt.style.use('seaborn-whitegrid')
25
26
27 # ***** Koordinaten des Observatory in folgender Form: *****
28 # longitude = 289.5967661 in Grad
29 # latitude = -24.62586593 in Grad
30 # altitude = 2635.43 in Meter
31
32 # ***** BITTE AUF EIGENE KOORDINATEN ÄNDERN *****
33 # Falls das versäumt wird sind die baryzentrischen Korrekturen falsch
34
35 # Koordinaten vom Berthold
36 # longitude = +7.4775
37 # latitude = 49.47527777778
38 # altitude = 200
39
40 # Koordinaten des Wise Observatory in Israel
41 # longitude = +34.76333333
42 # latitude = 30.59583333
43 # altitude = 875
44
45 # Koordinaten von Siegfried Hold
46 longitude = +15.68461111
47 latitude = 47.09161111
48 altitude = 380
49 # *****
50
51 # ***** BITTE AUF EIGENE KOORDINATEN ÄNDERN *****
52 # ***** Eingabe der Koordinaten des Sterns in folgender Form: *****
53 # Falls das versäumt wird sind die baryzentrischen Korrekturen falsch
54 # ra2000 = 030.20313477 in Grad
55 # dec2000 = -12.87498346 in Grad
56
57
58 # ra2000 = 030.20313477 in Grad
59 # dec2000 = -12.87498346 in Grad
60
61 # Koordinaten von del Cep
62 # ra2000 = 22.48618473
63 # dec2000 = +58.415198
64
65 # Koordinaten von gam Cya
66 # ra2000 = 385.55708
67 # dec2000 = +40.2566
68
69 # Koordinaten von Betageuze
70 # ra2000 = 88.7929583
71 # dec2000 = 7.40705555
72
73 # Koordinaten von thetal Ori C
74 # ra2000 = 83.81858333
75 # dec2000 = -5.389694444
76
77 # *****
78 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
79 files = input('Pfad und Name der Spektren (nutze wildcards): ')
80 filelist = glob.glob(files)
81
82 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
83 # zeitliche Ordnung
84 filelist.sort()
85
86 # Ausdruck der Liste
87 print('\nSpektrenliste: \n')
88 print('Anzahl der Spektren: ', len(filelist), '\n')
89
90 # Liste der wählbaren Linien:
91 Linien = {'CIV 726': 7726.2,
92          'CIII 7837': 7837.25,
93          'HeI 6678': 6678.149,
94          'FeI 6678': 6677.9865,
95          'FeI 6634': 6633.7492,
96          'FeI 6609': 6609.1098,
97          'H_alpha': 6562.817,
98          'FeI 6546': 6546.24,
99          'FeII 6516': 6516.0783,
100         'FeI 6463': 6462.725,
101         'FeII 6456': 6456.3805,
102         'FeI 6417': 6416.9386,
103         'FeI 6412': 6411.6592,
104         'FeI 6408': 6408.0272,
105         'FeI 6409': 6409.0008,
106         'FeI 6394': 6393.6009,
107         'SiII 6371': 6371.36,
108         'SiII 6347': 6347.10,
109         'FeI 6265': 6265.1335,
110         'FeI 6256': 6256.3611,
111

```

Figure 28.1: Skript zur automatischen Messung der RV einer wählbaren Linie, Minimumbestimmung per Regression.

- Spline
- Gaussfit

Normalerweise sind in fits-1d-Spektren-Dateien im Header das Beobachtungsdatum (als JD) angegeben. Zusammen mit dem Ort der Beobachtung (Koordinaten des Observatoriums) und den Koordinaten des Objekts besteht dann die Möglichkeit, die heliozentrische (oder baryzentrische) Korrektur innerhalb des Skripts zu berechnen und die aus den Wellenlängen des Linienminimums berechneten RV's direkt heliozentrisch zu korrigieren und auszugeben.

28.1. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels einer Regression

Verwendet wird das Skript *RV_MessungAnLinie_Zeitreihe_fits_perRegression.py* (Abb. 28.1 bis 28.3, die Zeilen 108 bis 210 (Linienliste) sind übersprungen).

Um das Skript zu verwenden müssen zuerst einige Anpassungen vorgenommen werden. Als erstes sind die geografischen Koordinaten des beobachtenden Observatorium (longitude, latitude in Grad, altitude in Meter) ab Zeile 46 eingegeben werden. Dann die Koordinaten des beobachteten Objekts ab Zeile 70 (ra2000 und dec2000 in Grad). Nach dem Speichern des so modifizierten Skripts kann es gestartet werden. Man wird nach Pfad und Namenstamm (wildcards benutzen !) der fits-Dateien gefragt. Dann nach der zu vermessenden Linie, die in der in der Konsole ausgedruckten Linienliste enthalten sein muss (falls das nicht der Fall ist, muss man vorab die Linienliste im Skript entsprechend erweitern). Dann wird noch die Systemgeschwindigkeit erfragt. Falls die nicht bekannt ist, einfach Null eingeben. Danach werden die einzelnen Spektren nacheinander ausgewertet und die ermittelten Daten in einer ascii-Datei abgespeichert (im tab-Format, Spalten 'Spektrum, JD, BC, RV, RV_bc und miniFlux', also Spektrumname, Julianisches Datum der Beobachtung, baryzentrische Korrektur, Radialgeschwindigkeit, baryzentrisch korrigierte Radialgeschwindigkeit und Flux des Minimums).

Das Skript ist nur geeignet für Linien, die ein eindeutiges Minimum aufweisen.

```
RV_MessungAnLinie_Zeitserie_fits_per...
~/Arbeitsordner/PythonPro.../Codeschnipsel/RV_...

211 HeI 4026: 4026.191
212 print('Linienauswahl: ', Linien.keys())
213
214 # Eingabe der zu messenden Linie:
215 Linie = input('Geben Sie den Namen der zu messenden Linie ein: ')
216 wellenlaenge = Linien[Linie]
217
218 # Eingabe der Systemgeschwindigkeit
219 systemgeschwindigkeit = float(
220     input('Geben Sie eine Systemgeschwindigkeit in km/s ein: '))
221 systemgeschwindigkeit = systemgeschwindigkeit / 299772 * wellenlaenge
222
223
224 # Abarbeiten der filelist, Einlesen von flux und header, Auswahl des Flux um
    die Linie:
225
226 # Definition von Variablen
227 hjd = np.zeros(len(filelist))
228 corr = np.zeros(len(filelist))
229 obs_time = np.zeros(len(filelist))
230 RV = np.zeros(len(filelist))
231 RV_bc = np.zeros(len(filelist))
232 miniflux = np.zeros(len(filelist))
233
234
235 for i in range(len(filelist)):
236     flux, header = fits.getdata(filelist[i], header=True)
237
238     if 'JD-OBS' in header:
239         obs_time[i] = float(header['JD-OBS'])
240     elif 'JD_OBS' in header:
241         obs_time[i] = float(header['JD_OBS'])
242     elif 'MJD-OBS' in header:
243         mjd = header['MJD-OBS']
244         obs_time[i] = mjd + 2400000.5
245     elif 'BAS_MJD' in header:
246         obs_time[i] = float(header['BAS_MJD'])
247     else:
248         print('Es ist kein Beobachtungszeitpunkt im Header')
249         break
250
251 # Berechnung der heliozentrischen Korrektur und Zeit:
252 corr[i], hjd[i] = pyasl.helcorr(Longitude, latitude, altitude,
253                               ra2000, dec2000, obs_time[i], debug=False)
254
255 print('\n*filelist[i]*:')
256 print('Beobachtungszeitpunkt: ', obs_time[i])
257 print('Barycentric correction [km/s]: ', corr[i])
258
259 if 'CRPIX1' in header:
260     refpix = int(header['CRPIX1'])
261 else:
262     refpix = 1
263
264 step = float(header['CDELTA1'])
265
```

```
Öffnen  RV_MessungAnLinie_Zeitserie_fits_per...
~/Arbeitsordner/PythonPro.../Codeschnipsel/RV_...

260
261     refpix = 1
262
263     step = float(header['CDELTA1'])
264
265     lambda0 = float(header['CRVAL1']) - (refpix - 1)
266
267     index_wellenlaenge = int(
268         (wellenlaenge - lambda0 + systemgeschwindigke
269
270     # *****
271     # Breite des Suchintervalls, Breite in Angström,
272     suchintervall = int(6/step)
273     # *****
274     intervallflux = np.zeros(suchintervall)
275     intervallwave = np.zeros(suchintervall)
276     for j in range(suchintervall):
277         intervallflux[j] = flux[j + index_wellenlaeng
278         intervallwave[j] = lambda0 + \
279             (j + index_wellenlaenge - int(suchinterva
280
281     linienminimum_wave_index = intervallflux.argmin()
282
283     # Das folgende Intervall anpassen, bei verrauscht
```

```

273 #
274 intervallflux = np.zeros(suchintervall)
275 intervallwave = np.zeros(suchintervall)
276 for j in range(suchintervall):
277     intervallflux[j] = flux[j + index_wellenlaenge - int(suchintervall/2)]
278     intervallwave[j] = lambda0 + \
279         (j + index_wellenlaenge - int(suchintervall/2))*step
280
281 linienminimum_wave_index = intervallflux.argmin()
282
283 # Das folgende Intervall anpassen, bei verrauschten Linien breit (-20,+21)
284 # bei nicht verrauschten, schmalen Linien klein (-2,+3)
285 linienminimum_intervall = np.arange(
286     linienminimum_wave_index-4, linienminimum_wave_index+5)
287
288 # Wellenlängen und Flux direkt um das Linienminimum
289 wl = intervallwave[linienminimum_intervall]
290 fl = intervallflux[linienminimum_intervall]
291
292 # Regression, Grad anpassen (2 oder 4 oder 6)
293 grad = 4
294 model = np.polyfit(wl, fl, grad)
295
296 polyline = np.linspace(wl[0], wl[-1], 50)
297 modflux = model(polyline)
298
299 # Linienminimum berechnen
300 miniflux[i] = modflux.min()
301 miniwave = polyline[modflux.argmin()]
302 miniwave_bc = miniwave*(1 + corr[i]/299792)
303
304 # Plotten
305 fig = plt.figure()
306 plt.plot(intervallwave, intervallflux, 'o-')
307 plt.plot(polyline, modflux)
308 plt.plot(miniwave, miniflux[i], 'o', color='black')
309
310 # RV nur bc-korrigiert, sysv nicht berücksichtigt:
311 RV_bc[i] = (miniwave_bc - wellenlaenge)/wellenlaenge*299792
312 print('baryzentrisch korrigierte RV: ', RV_bc[i])
313 # RV ohne baryzentrische Korrektur
314 RV[i] = (miniwave - wellenlaenge)/wellenlaenge*299792
315
316 # Plot der RV's
317 fig = plt.figure()
318 plt.plot(obs_time, RV, 'bo', markersize=1)
319
320 # Abspeichern als ascii-Datei
321 ascii.write([filelist, obs_time, corr, RV, RV_bc, miniflux], linie +
322     '_RV_Regression_grad'+str(grad)+'.dat', overwrite=True,
323     names=['Spektrum', 'JD', 'RV', 'RV_bc',
324     'miniflux'], format='tab')
325
326
327

```

Figure 28.3: Skript zur automatischen Messung der RV einer wählbaren Linie, Minimumbestimmung per Regression (Fortsetzung).

28.2. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels einer RBF

Das Skript `RV_MessungAnLinie_Zeitserie_fits_perRBF.py` verwendet statt einer Regression im Bereich des Linienminimum eine radial basis function (rbf). Der zentrale Code ist in Abb. 28.4 wieder gegeben. In Zeile 279 wird eine rbf aus den Wellenlängen/Fluxwerten des Minimumsuchintervalls gebildet, deren Rigidität mit einem Smoothingparameter 'smooth' gesteuert wird. In Zeile 281 wird das Minimumsuchintervall rebinned (10fach höhere Auflösung) und in Zeile 283 die rbf auf diese neue Intervall angewendet. Das Minimum des rbf-Modells wird in Zeile 288 berechnet. Die weiteren Schritte entsprechen dem Skript im vorhergehenden Abschnitt.

28.3. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels eines kubischen Splines

Die Modellierung des Minimumbereichs einer Linie mittels eines kubischen Splines ist im Skript `RV_MessungAnLinie_Zeitserie_fits_perSpline.py` verwirklicht. Das Skript unterscheidet sich von den beiden vorhergehenden nur im Kern (Abb. 28.5) ab Zeile 271.

Das Skript funktioniert auch mit völlig asymmetrischen Linien, auch mit blends. Es wird in jedem Fall das Minimum der Linie bestimmt. Die Berücksichtigung von lokalem Rauschen kann mit dem in Zeile 272 verwendeten Smoothingfaktor s unterdrückt werden.

28.4. Automatische Bestimmung der Radialgeschwindigkeit an einer Linie in einer Spektrenserie im fits-Format mittels eines Gauß-fittings

Eine Absorptionslinie kann allgemein mit einer angefitzten Gaußkurve modelliert werden. Das geschieht am besten mehrstufig. Zuerst wird die gesamte Linie als Gaußkurve modelliert, wobei das Minimum der Modellkurve deutlich vom wahren Minimum abweichen kann. Der Wellenlängenbereich wird dann in einer zweiten fitting-Aktion eingeeengt, wodurch nur noch der untere Teil der Linie modelliert wird.

```

269 # *****
270 intervallflux = np.zeros(suchintervall)
271 intervallwave = np.zeros(suchintervall)
272 for j in range(suchintervall):
273     intervallflux[j] = flux[j] + index_wellenlaenge - int(suchintervall/2))
274     intervallwave[j] = lambda0 + \
275         (j + index_wellenlaenge - int(suchintervall/2))*step
276
277 # Radial basis function (RBF) über das Spektrum im Intervall
278 # smooth anpassen, 0. = Funktion geht durch alle Punkte, >0. = ausgleich
279 rbf = Rbf(intervallwave, intervallflux, smooth=0.1)
280
281 newwaveintervall = np.arange(
282     intervallwave[0], intervallwave[-1], step/10)
283 newfluxinterpolated = rbf(newwaveintervall)
284 linienminimum_flux[i] = newfluxinterpolated.min()
285 # linienminimum_wave[i] = intervallwave[0] + \
286 #     newfluxinterpolated.argmax()*step/10
287 linienminimum_wave[i] = newwaveintervall[newfluxinterpolated.argmax()]
288 linienminimum_bc[i] = (linienminimum_wave[i]*1 + corr[i]/299792)
289 # RV nur bc-korrigiert, sysv nicht berücksichtigt:
290 RV_bc[i] = (linienminimum_bc[i] - wellenlaenge)/wellenlaenge*299792
291 RV[i] = (linienminimum_wave[i] - wellenlaenge)/wellenlaenge*299792
292 print('RV baryzentrisch korrigiert in km/s: ', RV_bc[i])
293
294 # Plotten
295 fig = plt.figure()
296 plt.plot(intervallwave, intervallflux, 'o')
297 plt.plot(newwaveintervall, newfluxinterpolated, '-')
298 plt.plot(linienminimum_wave[i], linienminimum_flux[i], 'or')
299
300 # Plot der RV's
301 fig = plt.figure()
302 fig = plt.plot(obs_time, RV, 'bo', markersize=1)
303 # plt.plot(obs_time, RV, 'bo', markersize=1)
304
305 # Abspeichern als ascii-Datei
306 ascii.write([filelist, obs_time, corr, RV, RV_bc], linie+'RV_perRBF'+'.dat',
307             overwrite=True,
308             names=['Spektrum', 'JD', 'BC', 'RV', 'RV_BC'], format='tab')
309
310 # ascii.write([obs_time, apex3], linie+'apex'+'.dat', overwrite=True,
311 #             names=['JD', 'apex'], format='tab')
312
313
314 # Speichern von obs time und corr in ascii-file
315 # ascii.write([obs_time, corr], linie+'Tabelle_obs_time_bc.dat', overwrite=True,
316 #             names=['JD', 'BARYCORR'], format='tab')
317
318 # Speichern von obs time und jhd in ascii-file
319 # ascii.write([obs_time, jhd], linie+'Tabelle_obs_time_jhd.dat', overwrite=True,
320 #             names=['JD', 'HJD'], format='tab')
321
322

```

Figure 28.4: Skript zur automatischen Messung der RV einer wählbaren Linie, Minimumbestimmung per RBF.

```

261 # Breite des Suchintervalls, Breite in Angström, anpassen.
262 suchintervall = int(4/step)
263 # *****
264 intervallflux = np.zeros(suchintervall)
265 intervallwave = np.zeros(suchintervall)
266 for j in range(suchintervall):
267     intervallflux[j] = flux[j] + index_wellenlaenge - int(suchintervall/2))
268     intervallwave[j] = lambda0 + \
269         (j + index_wellenlaenge - int(suchintervall/2))*step
270
271 # Spline über das Spektrum im Intervall
272 tck = interpolate.splrep(intervallwave, intervallflux, s=0.001)
273 newwaveintervall = np.arange(
274     intervallwave[0], intervallwave[-1], step/50)
275 newfluxinterpolated = interpolate.splev(newwaveintervall, tck, der=0)
276 linienminimum_flux[i] = newfluxinterpolated.min()
277 # linienminimum_wave[i] = intervallwave[0] + \
278 #     newfluxinterpolated.argmax()*step/10
279 linienminimum_wave[i] = newwaveintervall[newfluxinterpolated.argmax()]
280 linienminimum_bc[i] = (linienminimum_wave[i]*1 + corr[i]/299792)
281 # RV nur bc-korrigiert, sysv nicht berücksichtigt:
282 RV_bc[i] = (linienminimum_bc[i] - wellenlaenge)/wellenlaenge*299792
283 print('\nbaryzentrisch korrigierte RV: ', RV_bc[i])
284 RV[i] = (linienminimum_wave[i] - wellenlaenge)/wellenlaenge*299792
285
286 # Plotten
287 fig = plt.figure()
288 plt.plot(intervallwave, intervallflux, 'o')
289 plt.plot(newwaveintervall, newfluxinterpolated, '-')
290 plt.plot(linienminimum_wave[i], linienminimum_flux[i], 'o', color='black')
291
292 # Plot der RV's
293 fig = plt.figure()
294 fig = plt.plot(obs_time, RV, 'bo', markersize=1)
295 # plt.plot(obs_time, RV, 'bo', markersize=1)
296
297 # Abspeichern als ascii-Datei
298 ascii.write([filelist, obs_time, corr, RV, RV_bc, linienminimum_flux],
299             linie+'RV_perSpline'+'.dat', overwrite=True,
300             names=['Spektrum', 'JD', 'BC', 'RV', 'RV_BC',
301                 'RV_bc', 'linienminimum_flux'], format='tab')
302
303 # ascii.write([obs_time, apex3], linie+'apex'+'.dat', overwrite=True,
304 #             names=['JD', 'apex'], format='tab')
305
306 # Speichern von obs time und corr in ascii-file
307 # ascii.write([obs_time, corr], linie+'Tabelle_obs_time_bc.dat', overwrite=True,
308 #             names=['JD', 'BARYCORR'], format='tab')
309
310 # Speichern von obs time und jhd in ascii-file
311 # ascii.write([obs_time, jhd], linie+'Tabelle_obs_time_jhd.dat', overwrite=True,
312 #             names=['JD', 'HJD'], format='tab')
313

```

Figure 28.5: Skript zur automatischen Messung der RV einer wählbaren Linie, Minimumbestimmung per Spline.

Meist ist dann das Minimum des Modells bereits deckungsgleich mit dem Minimum der gemessenen Linie. Zur Sicherheit wird in einer dritten Stufe der Wellenlängenbereich auf das unmittelbare Umfeld des in der zweiten Stufe ermittelten Minimums begrenzt und der Flux per Gaußfitting modelliert. Die Wellenlänge dieses dritten Minimums ist dann das Ergebnis. Der mittlere Teil des Skripts (das mehrstufige Gaußfitting), das ansonsten mit den vorhergehenden Skripten gleich ist, ist in Abb. 28.6 abgebildet. Der sigma-Wert der Gaußfunktion (Zeile 288) muß an die reale Linienbreite angepasst werden, falls im ersten Anlauf das Gaußfitting mißlingt.

Nach den Erfahrungen des Autors ist das Skript den vorherigen Skripten (automatische RV-Bestimmung per Regression, per RBF oder per Spline) unterlegen. Das Gaußfitting der ersten Stufe versagt oft und die ermittelten Linienminima streuen stärker. Das letztere mag daran liegen, dass gemessene Absorptionslinien häufig asymmetrisch sind und per Gaußfit nur symmetrische Linien genau zu modellieren sind. Die Symmetrie der Linien ist für die anderen Modellierungsmethoden (SBF, Spline, Regression) keine Einschränkung.

Part X.

Bestimmung von Äquivalentweiten

29. Bestimmung der Äquivalentweite einer Linie in einer Spektrenserie im fits-Format

Die Messung der Äquivalentweite einer Absorptions- oder Emissionslinie in einem 1d-Spektrum oder einer Serie von diesen kann mit dem Pythonskript *Zeitreihe_EW.py* erfolgen (Code in Abb. 29.1). Der Integrationsbereich wird interaktiv per Mausclick im Spektrum definiert oder durch Eingabe der Wellenlängen für den Anfang und das Ende (Zeile 64). Die ermittelten EW's werden zusammen mit dem Spektrumbezeichner in einer ascii-Tabelle (tab-Format) gespeichert.

Part XI.

Kreuzkorrelationen (KK)

30. KK für 1d-Spektren des Dateityps fit

Möchte man die relative Verschiebung (beispielsweise wegen des Dopplereffekts) zweier überlappender Spektrenausschnitte bestimmen ist eine Methode dafür die Kreuzkorrelation. Dabei wird das „target“-Spektrum über dem „template“³² rechnerisch systematisch verschoben und die überlappende Fläche als Integral berechnet. Das ergibt die Kreuzkorrelationsfunktion, deren Maximum maximale Überlappung (Deckung) bedeutet und damit die wahrscheinlichste Dopplerverschiebung repräsentiert. Absorptionslinien, die in einem Spektrum vorhanden sind, im andern nicht³³, stören die Auswertung, weil sie die KK-Funktion verändern. Ungünstig sind auch terr. Linien wenn zu unterschiedlichen Zeitpunkten gemessene Spektren baryzentrisch korrigiert verglichen werden, in denen das Sternspektrum doppler-, aber die terrestrischen Absorptionslinien im Takt der baryzentrischen Korrektur verschoben sind. Deshalb nutzt man besser Spektrenausschnitte, die schwache oder keine terr. Linien enthalten, löscht störende Linien (ersetzt ihr Flux durch 1 = Kontinuum) oder die Spektren werden vorher von den terrestrischen Absorptionen möglichst weitgehend befreit („trocknen“).

Für die Bestimmung von Radialgeschwindigkeiten von Sternen ist eine baryzentrische Korrektur

³²Als template wird am besten ein theoretisches Spektrum verwendet, dessen Linien alle bei ihrer Laborwellenlänge erscheinen, für das also $RV = 0$ gilt.

³³z.B. terrestrische Linien sind in einem theoretischen, berechneten Spektrum nicht enthalten

```

265 # *****
266 # Breite des Suchintervalls, Breite in Angström, anpassen.
267 suchintervall1 = int(4/step)
268 # *****
269 intervallflux1 = np.zeros(suchintervall1)
270 intervallwave1 = np.zeros(suchintervall1)
271 for j in range(suchintervall1):
272     intervallflux1[j] = flux[j] +
273         index_wellenlaenge - int(suchintervall1/2)
274     intervallwave1[j] = lambda0 + \
275         (j + index_wellenlaenge - int(suchintervall1/2))*step
276
277 fig = plt.figure()
278 plt.plot(intervallwave1, intervallflux1, 'k:')
279
280 intervall1_flux_min = intervallflux1.min()
281 intervallwave1_min = intervallwave1[intervallflux1.argmax()]
282
283 gf1 = fuf.GaussFit1d()
284
285 # Schätzwerte für das Gaußfitting:
286 # A: - für Absorption, + für Emission
287 gf1['A'] = -(1 - intervall1_flux_min)
288 gf1['sig'] = 1.5 # auch der Wert muss an die Linienbreite angepasst werden
289 gf1['off'] = 1.
290 gf1['mu'] = intervallwave1_min
291 gf1['lin'] = 0.
292
293 gf1.thaw(['A', 'sig', 'off', 'mu'])
294 gf1.setRestriction({'A': [None, 0]})
295 gf1.fit(intervallwave1, intervallflux1)
296 # gf1.parameterSummary()
297
298 linienminimum1[i] = gf1['mu']
299 if linienminimum1[i] <= wellenlaenge - 4:
300     break
301 if linienminimum1[i] >= wellenlaenge + 4:
302     break
303 linienminimum1_bc[i] = linienminimum1[i] * \
304     (1 + corr[i]/299792) # heliozentrische Korrektur
305 RV1[i] = (linienminimum1_bc[i] - wellenlaenge)/linienminimum1_bc[i]*299792
306
307 plt.plot(intervallwave1, gf1.model, 'r--')
308
309 # zweiter Durchlauf, eingegrenzt auf Intervall 2*sigma
310 mu_index = int((gf1['mu'] - lambda0)/step + reflipx - 1)
311 suchintervall2 = 2 * abs(int(gf1['sig']/step))
312 intervallflux2 = np.zeros(suchintervall2)
313 intervallwave2 = np.zeros(suchintervall2)
314 for j in range(len(intervallflux2)):
315     intervallflux2[j] = flux[j + mu_index - suchintervall2//2]
316     intervallwave2[j] = lambda0 + (j + mu_index - suchintervall2//2)*step
317 gf2 = fuf.GaussFit1d()
318 gf2.setRestriction({'A': [None, 0]})

```

```

317 gf2 = fuf.GaussFit1d()
318 gf2.setRestriction({'A': [None, 0]})
319 gf2['A'] = gf1['A']
320 gf2['sig'] = gf1['sig']
321 gf2['off'] = gf1['off']
322 gf2['mu'] = gf1['mu']
323 gf2.thaw(['A', 'sig', 'off', 'mu'])
324 gf2.fit(intervallwave2, intervallflux2)
325 # gf2.parameterSummary()
326 linienminimum2[i] = gf2['mu']
327 if linienminimum2[i] <= wellenlaenge - 4:
328     break
329 if linienminimum2[i] >= wellenlaenge + 4:
330     break
331 linienminimum2_bc[i] = linienminimum2[i] * (1 + corr[i]/299792)
332 RV2[i] = (linienminimum2_bc[i] - wellenlaenge)/linienminimum2_bc[i]*299792
333
334 plt.plot(intervallwave2, gf2.model, 'b--')
335
336 RV_diff1[i] = RV2[i] - RV1[i]
337
338 # dritter Durchlauf, eingegrenzt auf Intervall 2*sigma
339 mu_index = int((gf2['mu'] - lambda0)/step + reflipx - 1)
340 suchintervall3 = abs(int(1.5*gf2['sig']/step))
341 intervallflux3 = np.zeros(suchintervall3)
342 intervallwave3 = np.zeros(suchintervall3)
343 for k in range(len(intervallflux3)):
344     intervallflux3[k] = flux[k + mu_index - suchintervall3//2]
345     intervallwave3[k] = lambda0 + (k + mu_index - suchintervall3//2)*step
346 gf3 = fuf.GaussFit1d()
347 gf3.setRestriction({'A': [None, 0]})
348 gf3['A'] = gf2['A']
349 gf3['sig'] = gf2['sig']
350 gf3['off'] = gf2['off']
351 gf3['mu'] = gf2['mu']
352 gf3.thaw(['A', 'sig', 'off', 'mu'])
353 gf3.fit(intervallwave3, intervallflux3)
354 gf3.parameterSummary()
355 linienminimum3[i] = gf3['mu']
356 if linienminimum3[i] <= wellenlaenge - 4:
357     break
358 if linienminimum3[i] >= wellenlaenge + 4:
359     break
360 apex3[i] = gf3.evaluate(gf3['mu'])
361 linienminimum3_bc[i] = linienminimum3[i] * (1 + corr[i]/299792)
362 RV3[i] = (linienminimum3_bc[i] - wellenlaenge)/linienminimum3_bc[i]*299792
363
364 plt.plot(intervallwave3, gf3.model, 'g--')
365 plt.title(filelist[i] + ' Beobachtungszeitpunkt ' + str(obs_time[i]))
366 plt.plot(linienminimum3[i], apex3[i], 'o', color='black')
367
368 RV_diff2[i] = RV3[i] - RV2[i]
369
370

```

```

348 gf3['A'] = gf2['A']
349 gf3['sig'] = gf2['sig']
350 gf3['off'] = gf2['off']
351 gf3['mu'] = gf2['mu']
352 gf3.thaw(['A', 'sig', 'off', 'mu'])
353 gf3.fit(intervallwave3, intervallflux3)
354 gf3.parameterSummary()
355 linienminimum3[i] = gf3['mu']
356 if linienminimum3[i] <= wellenlaenge - 4:
357     break
358 if linienminimum3[i] >= wellenlaenge + 4:
359     break
360 apex3[i] = gf3.evaluate(gf3['mu'])
361 linienminimum3_bc[i] = linienminimum3[i] * (1 + corr[i]/299792)
362 RV3[i] = (linienminimum3_bc[i] - wellenlaenge)/linienminimum3_bc[i]*299792
363
364 plt.plot(intervallwave3, gf3.model, 'g--')
365 plt.title(filelist[i] + ' Beobachtungszeitpunkt ' + str(obs_time[i]))
366 plt.plot(linienminimum3[i], apex3[i], 'o', color='black')
367
368 RV_diff2[i] = RV3[i] - RV2[i]
369
370
371 # Plot der RV's
372 fig=plt.figure()
373 plt.plot(obs_time, RV1, 'bo', markersize=1)
374 plt.plot(obs_time, RV2, 'r+', markersize=1)
375 plt.plot(obs_time, RV3, 'g+', markersize=1)
376
377 # Abspeichern als ascii-Datei
378 # ascii.write([obs_time, RV1], linie+'RV1'+'.dat', overwrite=True,
379 #             names=['JD', 'RV'], format='tab')
380
381 # ascii.write([obs_time, RV2], linie+'RV2'+'.dat', overwrite=True,
382 #             names=['JD', 'RV'], format='tab')
383
384 # ascii.write([obs_time, RV1, RV2, RV3], linie+'RV_perGaussfit'+'.dat', overwrite=True,
385 #             names=['JD', 'RV1', 'RV2', 'RV3'], format='tab')
386
387 # ascii.write([obs_time, apex3], linie+'apex'+'.dat', overwrite=True,
388 #             names=['JD', 'apex'], format='tab')
389
390 # ascii.write([obs_time, linienminimum3_bc],
391 #             linie+'heliozentrisch_korrigierte_Linienminima' +
392 #             '.dat', overwrite=True, names=['JD', 'WAVE'], format='tab')
393
394 # Speichern von obs time und corr in ascii-file
395 # ascii.write([obs_time, corr], linie+'Tabelle_obs_time_bc.dat', overwrite=True,
396 #             names=['JD', 'BARYCORR'], format='tab')
397
398 # Speichern von obs time und jhd in ascii-file
399 # ascii.write([obs_time, jhd], linie+'Tabelle_obs_time_hjd.dat', overwrite=True,
400 #             names=['JD', 'HJD'], format='tab')
401

```

Figure 28.6: Skript zur automatischen Messung der RV einer wählbaren Linie, Minimumbestimmung per Gaußfitting.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Berechnet für eine Zeitserie im fits-Format die Äquivalentweite einer Serie.
5 Eingabe der Integrationsgrenzen grafisch-iteraktiv oder manuell.
6 Die EW-Berechnung setzt voraus, dass für alle Spektren der Serie das gleiche
7 Wellenlängenintervall für die Berechnung des Integrals verwendet werden kann,
8 also keine wesentlichen RV-Änderungen stattfinden, oder wenn doch, dass die
9 Linie isoliert ist (also der Flux = 1 im Umfeld ist).
10
11 Created on Fri Oct 16 17:47:58 2020
12
13 @author: lothar
14 """
15
16 import numpy as np
17 from astropy.io import fits, ascii
18 import matplotlib.pyplot as plt
19 import glob
20
21 plt.ion()
22
23 # Fileliste erstellen. Spektren in einem (Unter)Ordner.
24 files = input("Pfad und Name der Spektren (nutze wildcards) : ")
25 filelist = glob.glob(files)
26
27 # Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
28 # zeitliche Ordnung
29 filelist.sort()
30
31 # Ausdruck der Liste
32 print("\nSpektralliste: \n")
33 print(filelist)
34 print("Anzahl der Spektren: ", len(filelist), "\n")
35
36 EW_dict = {}
37
38 k = 0
39 sp = fits.open(filelist[k], ignore_missing_end=True)
40 # print("\n\nHeader of the spectrum :\n\n", sp[0].header, "\n\n")
41
42 # Generation of arrays with the wavelengths and fluxes of the spectrum
43 flux = np.array(sp[0].data)
44 wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
45
46 for i in np.arange(sp[0].header["NAXIS1"]):
47     wave[i] = (
48         sp[0].header["CRVAL1"]
49         + (i - sp[0].header["CRPIX1"] + 1) * sp[0].header["CDELTA1"]
50     )
51     # The list wave contains the wavelengths of the pixels.
52     # Close the fits-file
53 sp.close()
54
55 # Plot the spectrum

```

```

56 with plt.ion():
57     fig = plt.figure(figsize=(10, 8))
58     plt.plot(wave, flux)
59     plt.xlabel("Wavelength [Angström]")
60     plt.ylabel("ADU")
61     plt.title("Spektrum " + filelist[0])
62     plt.grid(True)
63
64 fragel = input(
65     "Möchten Sie die Integrationsgrenzen zahlenmäßig eingeben oder \
66 per Mausclick (grafisch)? Geben Sie m oder g ein: "
67 )
68
69 if fragel == "g":
70     # Interaktives Festlegen der Integrationsgrenzen
71     # Falls die Darstellung des Spektrums interaktiv vergrößert wird die Eingabepunkte
72     # jeweils mit der rechten Maustaste löschen, bis dann wirklich die linke
73     # Seite der zu messenden Linie angeklickt wird.
74     pts = []
75     pts = np.asarray(plt.ginput(n=2, timeout=-1))
76     plt.plot(pts[:, 0], pts[:, 1], "o", markersize=3)
77     begin = pts[0, 0]
78     end = pts[1, 0]
79     print("Gewählter Integrationsbereich:", begin, " bis", end)
80     print()
81
82 if fragel == "m":
83     # Eingabe der Integrationsgrenzen
84     begin = float(
85         input("Geben Sie die kurzwellige Wellenlänge-Integrationsgrenze ein: ")
86     )
87     end = float(input("Geben Sie die langwellige Wellenlänge-Integrationsgrenze ein:
88 ))
89     print()
90 EW = np.zeros(len(filelist))
91
92 # Abarbeiten der filelist
93 for k in np.arange(len(filelist)):
94     sp = fits.open(filelist[k], ignore_missing_end=True)
95
96     # Generation of arrays with the wavelengths and fluxes of the spectrum
97     flux = np.array(sp[0].data)
98     wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
99
100     for i in np.arange(sp[0].header["NAXIS1"]):
101         wave[i] = (
102             sp[0].header["CRVAL1"]
103             + (i - sp[0].header["CRPIX1"] + 1) * sp[0].header["CDELTA1"]
104         )
105     # The list wave contains the wavelengths of the pixels.
106     # In the list flux the corresponding intensities.

```

```

87     end = float(input("Geben Sie die langwellige Wellenlänge-Integrationsgrenze ein:
88 ))
89     print()
90 EW = np.zeros(len(filelist))
91
92 # Abarbeiten der filelist
93 for k in np.arange(len(filelist)):
94     sp = fits.open(filelist[k], ignore_missing_end=True)
95
96     # Generation of arrays with the wavelengths and fluxes of the spectrum
97     flux = np.array(sp[0].data)
98     wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
99
100     for i in np.arange(sp[0].header["NAXIS1"]):
101         wave[i] = (
102             sp[0].header["CRVAL1"]
103             + (i - sp[0].header["CRPIX1"] + 1) * sp[0].header["CDELTA1"]
104         )
105     # The list wave contains the wavelengths of the pixels.
106     # In the list flux the corresponding intensities.
107     # Close the fits-file
108     sp.close()
109
110     for n in np.arange(len(wave)):
111         if wave[n] <= begin and wave[n + 1] > begin:
112             begin_n = n
113             begin_wave = wave[n]
114             begin_flux = np.median(flux[n - 2 : n + 2])
115             break
116     for n in np.arange(len(wave)):
117         if wave[n] <= end and wave[n + 1] > end:
118             end_n = n
119             end_wave = wave[n]
120             end_flux = np.median(flux[n - 2 : n + 2])
121             break
122     faktor = (end_flux - begin_flux) / (end_n - begin_n)
123     ew = 0
124     for m in np.arange(end_n - begin_n):
125         dif = begin_flux + m * faktor - flux[begin_n + m]
126         ew += dif * sp[0].header["CDELTA1"]
127     EW[k] = ew
128
129 print(filelist[k], " EW =", EW[k])
130 # Daten zum Weiterverarbeiten:
131 EW_dict[filelist[k]] = (begin_wave, end_wave, begin_flux, end_flux, EW[k])
132
133 ascii.write(
134     [filelist, EW],
135     files + str(int(begin)) + "_" + str(int(end)),
136     names=["Spektrum", "EW"],
137     overwrite=True,
138     format="tab",
139 )

```

Figure 29.1: Pythoncode des Skripts Zeitserie_EW.py zur Messung der Äquivalentweite einer Linie in einer Spektrenserie.

```

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe
Offnen [ ] Speichern

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Kreuzkorrelation
5abgeleitet von einem Beispiel in PyAstronomy
6https://www.hs-uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyaslDoc/aslDoc/
7crosscorr.html
8
9Es wird eine Kreuzkorrelation eines target-Spektrums bzgl. eines
10template-Spektrums durchgeführt. Beide liegen als fits vor.
11
12Stand 20180815
13
14@author: Lothar Schanne
15"""
16
17from future import print function, division
18from PyAstronomy import pyasl
19import numpy as np
20import matplotlib.pyplot as plt
21from astropy.io import fits
22
23
24obj = input('Geben Sie die Überschrift für die Grafiken ein: ')
25
26# Template auswählen
27# Pfad und Name des templates
28tfile = input('Pfad und Datei-Bezeichnung des template eingeben: ')
29# Einlesen von Header und Daten (Flux vom template in tf,
30# Header des template in theader gespeichert)
31tf, theader = fits.getdata(tfile, header=True)
32print('Minimum und Maximum im Template [ADU]: ', tf.min(), ' ', tf.max())
33# Header-Check Template
34theadr_flag = input('Möchten Sie den header des template komplett sehen? \n
35                      y/n:')
36if theader_flag == 'y':
37    print('Headerdaten des template: ')
38    print(theader)
39# Prüfung auf die nötigen header-Einträge
40print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
41if 'NAXIS' in theader:
42    print('Dimension, NAXIS: ', theader['NAXIS'])
43else:
44    print('Das ist kein 1d-Spektrum !')
45if 'NAXIS1' in theader:
46    tmax = theader['NAXIS1']
47    print('Anzahl der Werte (Abszisse), NAXIS1: ', tmax)
48else:
49    print('NAXIS1 fehlt im header !')
50if 'CRVAL1' in theader:
51    tcrval = theader['CRVAL1']
52    print('Anfangs-Wellenlänge, CRVAL1: ', tcrval)
53else:
54    print('CRVAL1 fehlt im header !')
55if 'CDELTA1' in theader:
56    tcddel = theader['CDELTA1']
57    print('Schrittweite der Wellenlänge, CDELTA1: ', tcddel)
58else:
59    print('CDELTA1 fehlt im header !')
60# Erzeugen eines numpy-Arrays mit den Wellenlängen des template
61tw = np.ones(tmax, dtype=float)
62for i in range(tmax):
63    tw[i] = tcrval + i*tcddel
64
65
66# Zu korrelierendes Spektrum (target) auswählen
67# Pfad und Name des target
68file = input('Pfad und Datei-Bezeichnung des target-Spektrums eingeben: ')
69# Einlesen von Header und Daten (Flux vom template in tf,
70# Header des template in theader gespeichert)
71f, header = fits.getdata(file, header=True)

```

Figure 30.1: Kreuzkorrelationskript für 1d-fits-Spektren.

nach VIII für alle beteiligten Spektren erforderlich.

In 30.1 ist das Listing eines Pythonskripts *CrossCorrelation_fits.py* wiedergegeben, das ein Targetspektrum über einem Templatespektrum (beide 1d als fits) kreuzkorreliert. Dabei sollte das Template einen etwas größeren Wellenlängenbereich abdecken wie das Target (damit links und rechts ausreichend Pixel für den Verschiebungsbereich zur Verfügung stehen).

Das Skript fragt nach einer gemeinsamen Überschrift für die später erzeugten Grafiken (z.B. soll das Objekt bezeichnet werden). Danach wird das Template ausgewählt (Pfad und Bezeichner), minimale und maximale Intensität ausgegeben³⁴ und der Header auf die notwendigen Parameter überprüft. Das gleiche wird anschließend mit dem Target durchgeführt. Ein Overplot beider Spektren zeigt dann grafisch wie beide Spektren aussehen. Die Kreuzkorrelation wird dann ab Zeile 121 durchgeführt. Die darin verwendeten Parameter für den Verschiebungsbereich (in km/s) und die Schrittweite der Verschiebung sowie die Anzahl der Datenpunkte an den Spektrenenden, die unberücksichtigt bleiben, sollten im Skript an den jeweiligen Fall angepasst werden. Das Skript weist dann das Maximum der Kreuzkorrelationsfunktion (die Radialgeschwindigkeit in km/s) und seine Richtung (rot- oder blauverschoben) in der Konsole aus und plottet die Kreuzkorrelationsfunktion mit einer Rotmarkierung des Maximums.

³⁴Zur Kontrolle, ob beide Spektren vergleichbaren Flux haben, was für eine ausgeprägtes Maximum der Kreuzkorrelationsfunktion wichtig ist. Normalerweise werden auf das Kontinuum normierte Spektren kreuzkorreliert. Das muss aber nicht sein. Auch nicht normierte Spektren können kreuzkorreliert werden, wenn sie vergleichbaren Fluxverlauf haben.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Kreuzkorrelation
5 abgeleitet von einem Beispiel in PyAstronomy
6 https://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyasDoc/aslDoc/
7 crosscorr.html
8
9 Es wird eine Kreuzkorrelation eines target-Spektrums bzgl. eines
10 template-Spektrums durchgeführt. Beide liegen als fits vor.
11
12 Stand 20180815
13
14 @author: Lothar Schanne
15 """
16
17 from __future__ import print function, division
18 from PyAstronomy import pyasl
19 import numpy as np
20 import matplotlib.pyplot as plt
21 from astropy.io import fits
22
23
24 obj = input('Geben Sie die Überschrift für die Grafiken ein: ')
25
26 # Template auswählen
27 # Pfad und Name des templates
28 tfile = input('Pfad und Dateizeichnung des template eingeben: ')
29 # Einlesen von Header und Daten (Flux vom template in tf,
30 # Header des template in theader gespeichert)
31 tf, theader = fits.getdata(tfile, header=True)
32 print('Minimum und Maximum im Template [ADU]: ', tf.min(), ' ', tf.max())
33 # Header-Check Template
34 theader_flag = input('Möchten Sie den header des template komplett sehen? \
35 j/n:')
36 if theader_flag == 'j':
37     print('Headerdaten des template:')
38     print(theader)
39 # Prüfung auf die nötigen header-Einträge
40 print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
41 if 'NAXIS' in theader:
42     print('Dimension, NAXIS: ', theader['NAXIS'])
43 else:
44     print('Das ist kein 1d-Spektrum !')
45 if 'NAXIS1' in theader:
46     tnax = theader['NAXIS1']
47     print('Anzahl der Werte (Abszisse), NAXIS1: ', tnax)
48 else:
49     print('NAXIS1 fehlt im header !')
50 if 'CRVAL1' in theader:
51     tcrval = theader['CRVAL1']
52     print('Anfangs-Wellenlänge, CRVAL1: ', tcrval)
53 else:
54     print('CRVAL1 fehlt im header !')
55 if 'CDEL1' in theader:
56     tcdel = theader['CDEL1']
57     print('Schrittweite der Wellenlänge, CDEL1: ', tcdel)
58 else:
59     print('CDEL1 fehlt im header !')
60 # Erzeugen eines numpy-Arrays mit den Wellenlängen des template
61 tw = np.ones(tnax, dtype=float)
62 for i in range(tnax):
63     tw[i] = tcrval + i*tcdel
64
65
66 # Zu korrelierendes Spektrum (target) auswählen
67 # Pfad und Name des target
68 file = input('Pfad und Dateizeichnung des target-Spektrums eingeben: ')
69 # Einlesen von Header und Daten (Flux vom template in tf,
70 # Header des template in theader gespeichert)
71 f, header = fits.getdata(file, header=True)
72 print('Minimum und Maximum im target [ADU]: ', f.min(), ' ', f.max())
73 # Header-Check target
74 header_flag = input('Möchten Sie den header des target komplett sehen? j/n:')
75 if header_flag == 'j':
76     print('Headerdaten des template:')
77     print(header)
78 # Prüfung auf die nötigen header-Einträge
79 print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
80 if 'NAXIS' in header:
81     print('Dimension, NAXIS: ', header['NAXIS'])
82 else:
83     print('Das ist kein 1d-Spektrum !')
84 if 'NAXIS1' in header:
85     nax = header['NAXIS1']
86     print('Anzahl der Werte (Abszisse), NAXIS1: ', nax)
87 else:
88     print('NAXIS1 fehlt im header !')
89 if 'CRVAL1' in header:
90     crval = header['CRVAL1']
91     print('Anfangs-Wellenlänge, CRVAL1: ', crval)
92 else:
93     print('CRVAL1 fehlt im header !')
94 if 'CDEL1' in header:
95     cdel = header['CDEL1']
96     print('Schrittweite der Wellenlänge, CDEL1: ', cdel)
97 else:
98     print('CDEL1 fehlt im header !')
99 # Erzeugen eines numpy-Arrays mit den Wellenlängen des target
100 w = np.ones(nax, dtype=float)
101 for i in range(nax):
102     w[i] = crval + i*cdel
103
104
105 # Plot template und target
106 plt.title(obj+'-Template (blau) and Targetspektrum (rot)')
107 plt.plot(tw, tf, 'b-')
108 plt.plot(w, f, 'r-')
109 plt.grid(True)
110 plt.xlabel('Wellenlänge [Angström]')
111 plt.savefig(obj+'.png')
112 plt.savefig(obj+'.pdf')
113 plt.show()
114
115 # Cross-Correlation ausführen.
116 # Das RV-range ist (Parameter 1) - bis (Parameter 2) km/s in
117 # Schritten von (Parameter 3) km/s.
118 # Die ersten und letzten (Parameter 4) Datenpunkte sind unberücksichtigt.
119 # muss angepasst werden, damit die Spektren noch überlappen können bei der
120 # maximalen Verschiebung (Parameter 2)
121 rv, cc = pyasl.crosscorrRV(w, f, tw, tf, -50., 50., .3, skipedge=80)
122
123 # Maximum der cross-correlation function
124 maxind = np.argmax(cc)
125
126 print("Die Cross-correlation function ist maximal bei dRV = ", rv[maxind],
127       " km/s.")
128 if rv[maxind] > 0.0:
129     print("Rotverschiebung des target gegenüber dem template.")
130 else:
131     print("Blauverschiebung des target gegenüber dem template")
132
133 # Plot CroCo-Funktion
134 plt.plot(rv, cc, 'bp-')
135 plt.plot(rv[maxind], cc[maxind], 'ro')
136 plt.title(obj+' : Kreuzkorrelationsfunktion')
137 plt.grid(True)
138 plt.xlabel('km/s')
139 plt.savefig(obj+'_CroCo.png')
140 plt.savefig(obj+'_CroCo.pdf')
141 plt.show()

```

Figure 31.1: Kreuzkorrelationsskript für 1d-ascii-Spektren.

31. KK für 1d-Spektren im ASCII-Format

Analog zu 30.1 sind im Skript *CrossCorrelation_fuer_ascii_files.py* KK für Target und Template im ascii-Format programmiert, wobei diese zwei Spalten ohne Überschrift enthalten mit den Wellenlängen und den Intensitäten, jeweils im float-Format (Dezimalzahlen mit Punkt als Dezimalzeichen). Das Trennzeichen (z.B. Komma im Falle von csv-Dateien) wird abgefragt.

32. KK einer Spektrenserie im fits-Format mit einem Template

Die Durchführung von Kreuzkorrelationen zwischen einem Template und einer Serie von fits-1d-Spektren erfolgt in dem Pythonskript *CrossCorrelation_fits_Serie.py*. Der Code ist in Abb. 32.1 zu finden. Die ermittelten RV's werden in einer ascii-Datei namens *RV_Liste_* gespeichert.

33. KK einer Spektrenserie im ascii-Format mit einem Template

Analog zum vorherigen Skript werden im Code *CrossCorrelation_dat_Serie.py* Spektren im ascii-Format (als .dat vorliegend) mit einem Template gekreuzkorreliert.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Kreuzkorrelation
5 abgeleitet von einem Beispiel in PyAstronomy
6 https://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyasDoc/aslDoc/
7 crosscorr.html
8
9 Es wird eine Kreuzkorrelation einer Serie von target-Spektren bzgl. eines
10 template-Spektrums durchgeführt. Beide liegen als fits vor.
11
12 Stand 20180815
13
14 @author: Lothar Schanne
15 """
16
17
18 from PyAstronomy import pyasl
19 import numpy as np
20 import matplotlib.pyplot as plt
21 from astropy.io import fits
22 from astropy.io import ascii
23 from astropy.table import Table
24 import glob
25
26
27 obj = input('Geben Sie die Überschrift für die Grafiken ein: ')
28
29 # Template auswählen
30 # Pfad und Name des templates
31 tfile = input('Pfad und Dateizeichnung des template eingeben: ')
32
33 # Einlesen von Header und Daten (Flux vom template in tf,
34 # Header des template in theader gespeichert)
35 tf, theader = fits.getdata(tfile, header=True)
36 print('Minimum und Maximum im Template [ADU]: ', tf.min(), ' ', tf.max())
37
38 tmax = theader['MAXIS1']
39 tcrval = theader['CRVAL1']
40 tcdel = theader['CDEL1']
41
42 tw = np.ones(tmax, dtype=float)

```

```

41
42 tw = np.ones(tmax, dtype=float)
43 tcrval = tcrval + (1 - theader['CRPIX1']) * tcdel
44 for i in range(tmax):
45     tw[i] = tcrval + i * tcdel
46
47
48 # Zu korrelierendes Spektrum (target) auswählen
49 # Pfad und Name des target
50 # Create file list. Spectra in a (sub)folder.
51 files = input('Path and name of the target spectra (use wildcards): ')
52 filelist = glob.glob(files)
53
54 # Sort alphabetically. If the spectrum files are named correctly, this results # in a
55 # temporal order.
56 filelist.sort()
57 # Printout of the list for control purposes.
58 print('\nList of target spectra:')
59 print(filelist)
60 print('\nNumber of target spectra: ', len(filelist), '\n')
61 print('Bitte warten. Berechnung läuft.')
62
63 RV = np.zeros(len(filelist))
64 jd = np.zeros(len(filelist))
65
66 for i in range(len(filelist)):
67     f, header = fits.getdata(filelist[i], header=True)
68     # print('Minimum und Maximum im'+filelist[i]+' [ADU]: ', f.min(), ' ', f.max())
69     max = header['MAXIS1']
70     crval = header['CRVAL1']
71     cdel = header['CDEL1']
72     if 'JD-OBS' in header:
73         jd[i] = header['JD-OBS']
74     else:
75         if 'JD-OBS' in header:
76             jd[i] = header['JD-OBS']
77         else:
78             if 'JD' in header:
79                 jd[i] = header['JD']
80
81 # Erzeugen eines numpy-Arrays mit den Wellenlängen des target
82 w = np.ones(max, dtype=float)
83 crval = crval + (1 - header['CRPIX1']) * cdel
84 for j in range(max):
85     w[j] = crval + j * cdel
86
87 # Cross-Correlation ausführen.
88 # Das RV-range ist (Parameter 1) - bis (Parameter 2) km/s in
89 # Schritten von (Parameter 3) km/s.
90 # Die ersten und letzten (Parameter 4) Datenpunkte sind unberücksichtigt.
91 # Muss angepasst werden, damit die Spektren noch überlappen können bei der
92 # maximalen Verschiebung (Parameter 2)
93 rv, cc = pyasl.crosscorrRV(
94     w, f, tw, tf, -100., 100., .5, mode='doppler', skipedge=20)
95
96 # Maximum der cross-correlation function finden
97 maxind = np.argmax(cc)
98 RV[i] = rv[maxind]
99
100 # Plot CroCo-Funktion
101 fig = plt.figure()
102 plt.plot(rv, cc, 'b-')
103 plt.plot(rv[maxind], cc[maxind], 'ro')
104 plt.title(obj+' : Kreuzkorrelationsfunktion '+filelist[i])
105 plt.grid(True)
106 plt.xlabel('km/s')
107 # plt.savefig(filelist[i]+'_CroCo.png')
108
109 print('Spektrum ', filelist[i], ' [km/s]')
110 for i in range(len(filelist)):
111     print(filelist[i], jd[i], RV[i])
112
113 data = Table([filelist, jd, RV], names=['Spektrum', 'JD-OBS', 'RV'])
114 ascii.write(data, 'RV_Liste'+file+'.dat', overwrite=True, format='tab')
115
116 plt.show()
117 print('Ende des Programs')

```

```

77
78     else:
79         if 'JD' in header:
80             jd[i] = header['JD']
81
82 # Erzeugen eines numpy-Arrays mit den Wellenlängen des target
83 w = np.ones(max, dtype=float)
84 crval = crval + (1 - header['CRPIX1']) * cdel
85 for j in range(max):
86     w[j] = crval + j * cdel
87
88 # Cross-Correlation ausführen.
89 # Das RV-range ist (Parameter 1) - bis (Parameter 2) km/s in
90 # Schritten von (Parameter 3) km/s.
91 # Die ersten und letzten (Parameter 4) Datenpunkte sind unberücksichtigt.
92 # Muss angepasst werden, damit die Spektren noch überlappen können bei der
93 # maximalen Verschiebung (Parameter 2)
94 rv, cc = pyasl.crosscorrRV(
95     w, f, tw, tf, -100., 100., .5, mode='doppler', skipedge=20)
96
97 # Maximum der cross-correlation function finden
98 maxind = np.argmax(cc)
99 RV[i] = rv[maxind]
100
101 # Plot CroCo-Funktion
102 fig = plt.figure()
103 plt.plot(rv, cc, 'b-')
104 plt.plot(rv[maxind], cc[maxind], 'ro')
105 plt.grid(True)
106 plt.xlabel('km/s')
107 # plt.savefig(filelist[i]+'_CroCo.png')
108
109 print('Spektrum ', filelist[i], ' [km/s]')
110 for i in range(len(filelist)):
111     print(filelist[i], jd[i], RV[i])
112
113 data = Table([filelist, jd, RV], names=['Spektrum', 'JD-OBS', 'RV'])
114 ascii.write(data, 'RV_Liste'+file+'.dat', overwrite=True, format='tab')
115
116 plt.show()
117 print('Ende des Programs')

```

Figure 32.1: Kreuzkorrelationskript für eine Spektrenserie im fits-Format.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Kreuzkorrelation
5 abgeleitet von einem Beispiel in PyAstronomy
6 https://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyaslDoc/aslDoc/
7 crosscorr.html
8
9 Es wird eine Kreuzkorrelation einer Serie von target-Spektren bzgl. eines
10 template-Spektrums durchgeführt. Beide liegen als dat vor.
11
12 Stand 20211225
13
14 @author: Lothar Schanne
15 """
16
17
18 from PyAstronomy import pyasl
19 import numpy as np
20 import matplotlib.pyplot as plt
21 from astropy.io import ascii
22 from astropy.table import Table
23 import glob
24 from linetools.spectra.xspectrumid import XSpectrumID
25
26
27 obj = input("Geben Sie die Überschrift für die Grafiken ein (ohne blank): ")
28
29 # Template auswählen
30 # Pfad und Name des templates
31 template_name = input("Pfad und Filebezeichnung des template eingeben: ")
32 template = np.loadtxt(template_name, skiprows=1)
33 print("Minimum und Maximum im Template [ADU]: ", template.min(), " ", template.max())
34
35 w = template[:, 0]
36 f = template[:, 1]
37
38 dt = (w[-1] - w[0]) / len(w)
39
40 template_newdata, dt_template = pyasl.binningx0dt(
41     w, f, dt=dt, x0=w[0], useBinCenter=True
42 )

```

```

41     w, f, dt=dt, x0=w[0], useBinCenter=True
42 )
43
44 # print("Template Beginn und Ende: ", template_newdata[0, 0], template_newdata[-1, 0])
45 print("Template Beginn und Ende: ", w[0], w[-1])
46
47 # Zu korrelierendes Spektrum (target) auswählen
48 # Pfad und Name des target
49 # Create file list. Spectra in a (sub)folder.
50 files = input("Path and name of the target spectra (use wildcards): ")
51 filelist = glob.glob(files)
52
53 # Sort alphabetically. If the spectrum files are named correctly, this results # in a
54 temporal order.
55
56 # Printout of the list for control purposes.
57 print("\nList of target spectra:")
58 print(filelist)
59 print("\nNumber of target spectra: ", len(filelist), "\n")
60 print("Bitte warten. Berechnung läuft")
61
62 RV = np.zeros(len(filelist))
63 jd = np.zeros(len(filelist))
64
65 for i in range(len(filelist)):
66     target = np.loadtxt(filelist[i], skiprows=1)
67     # print("Minimum und Maximum im+filelist[i]+ [ADU]: ', f.min(), ' ', f.max())
68
69     # Erzeugen je eines numpy-Arrays mit den Wellenlängen und Flux des target
70     tw = target[:, 0]
71     tf = target[:, 1]
72     binnumber = int((tw[-1] - tw[0]) / dt_template)
73
74     target_newdata, dt_target = pyasl.binningx0dt(
75         tw, tf, x0=tw[0], nbins=binnumber, useBinCenter=True
76     )
77
78     print(filelist[i])
79     print("Target Beginn und Ende:", target_newdata[0, 0], target_newdata[-1, 0])
80

```

```

80
81 # Cross-Correlation ausführen.
82 # Das RV-range list (Parameter 1) - bis (Parameter 2) km/s in
83 # Schritten von (Parameter 3) km/s.
84 # Die ersten und letzten (Parameter 4) Datenpunkte sind unberücksichtigt.
85 # Muss angepasst werden, damit die Spektren noch überlappen können bei der
86 # maximalen Verschiebung (Parameter 2)
87 rv, cc = pyasl.crosscorrRV(
88     target_newdata[:, 0],
89     target_newdata[:, 1],
90     w,
91     f,
92     -20.0,
93     20.0,
94     0.1,
95     modes="doppler",
96     skipedge=100,
97 )
98
99 # Maximum der cross-correlation function finden
100 maxind = np.argmax(cc)
101 RV[i] = rv[maxind]
102
103 # Plot CroCo-Funktion
104 fig = plt.figure()
105 plt.plot(rv, cc, "b-")
106 plt.plot(rv[maxind], cc[maxind], "ro")
107 plt.title(obj + " : Kreuzkorrelationsfunktion " + filelist[i])
108 plt.grid(True)
109 plt.xlabel("km/s")
110 # plt.savefig(filelist[i]+'_CroCo.png')
111
112 print("Spektrum ", "RV [km/s]")
113 for i in range(len(filelist)):
114     print(filelist[i], jd[i], RV[i])
115
116 data = Table([filelist, RV], names=["Spektrum", "RV"])
117 ascii.write(data, "RV_Liste_" + obj + ".dat", overwrite=True, format="tab")
118
119 plt.show()
120 print("Ende des Programms")

```

Figure 33.1: Kreuzkorrelationsskript für eine Spektrenserie im ascii-Format.