

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. Rebinning 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.rspec-astro.com/>

⁶http://aesefas.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 Instalierbarkeit. 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.

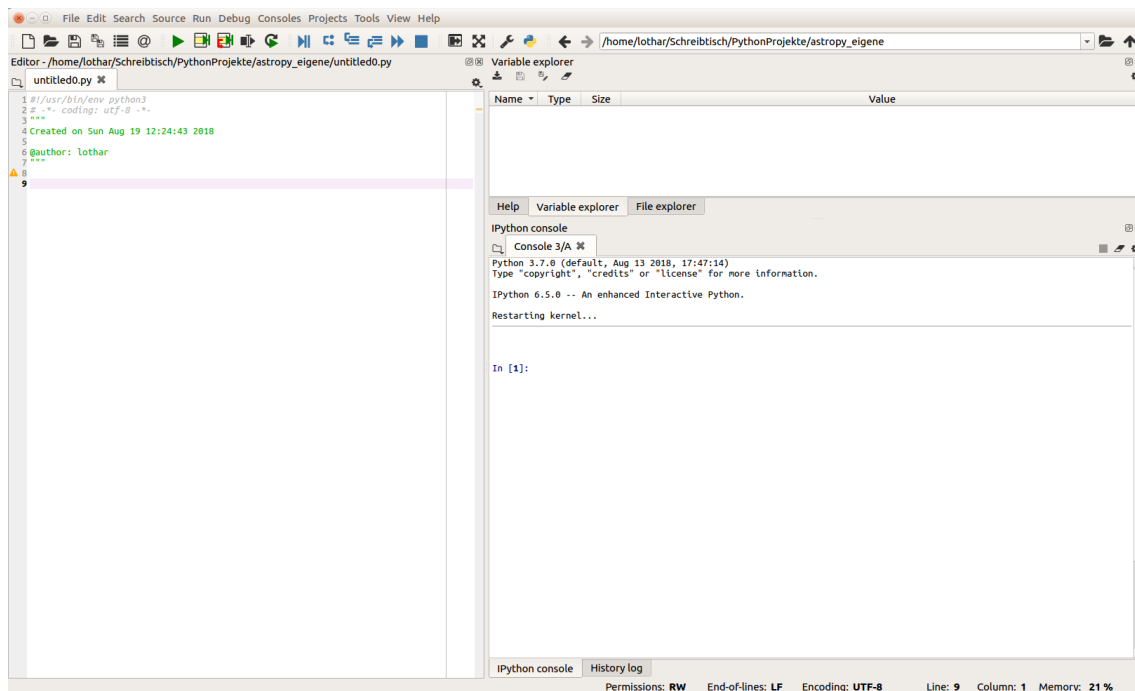


Figure 4.1: Spyder-Fenster nach dem Start.

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/>).

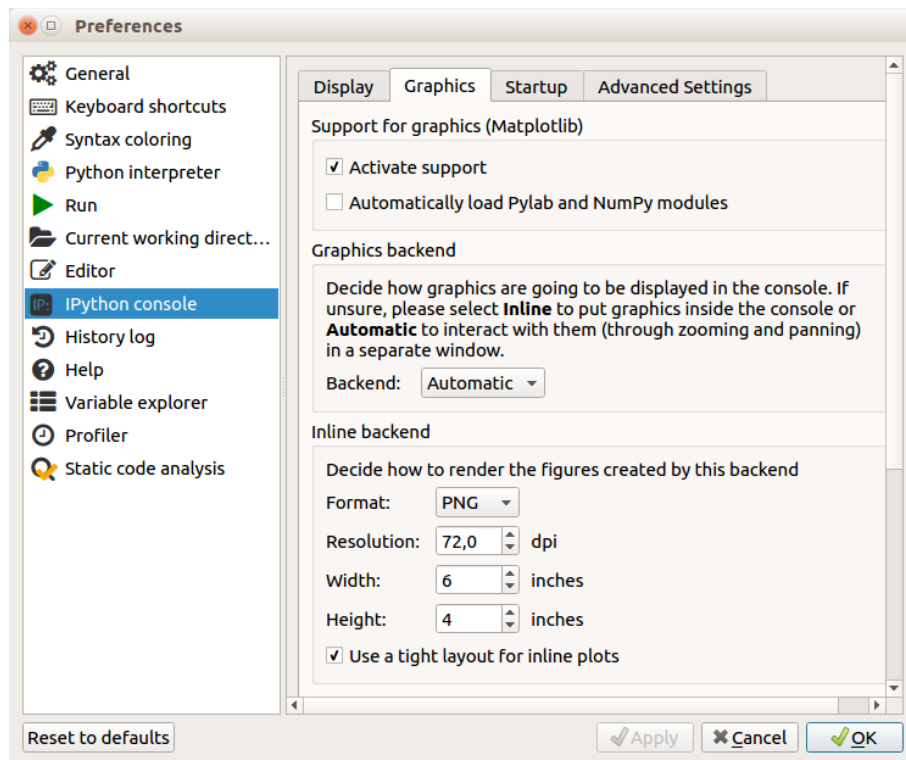


Figure 4.2: Einstellungen für die IPython-Konsole in Spyder.

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 einfaches 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/>

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*).

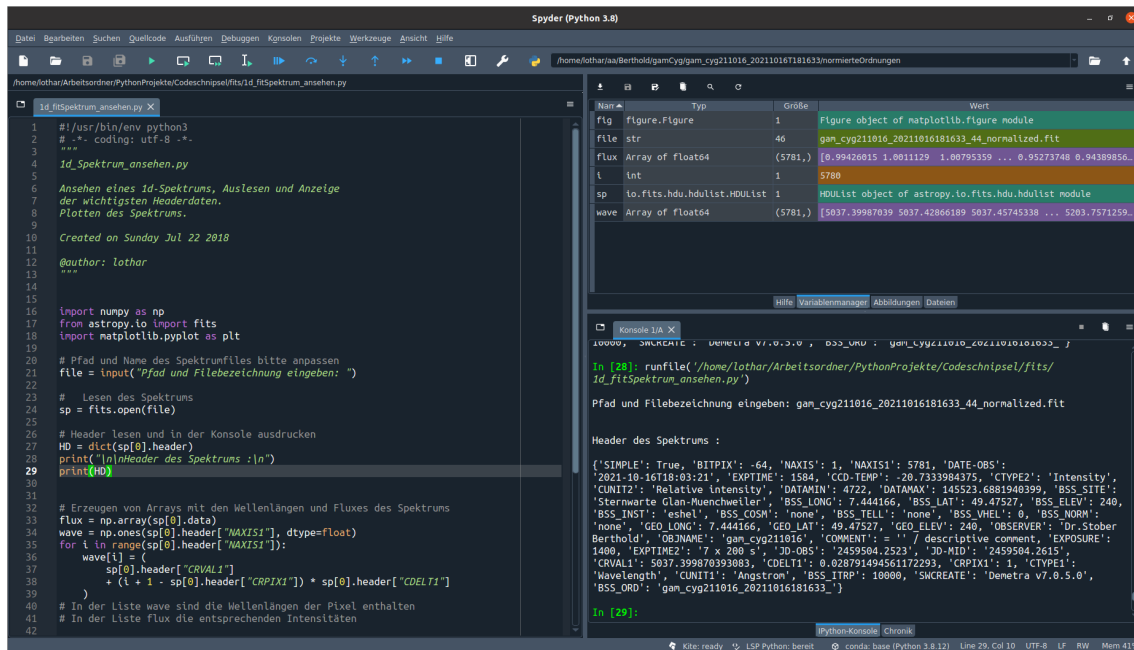


Figure 6.2: Ausdruck des Headers in der Konsole nach Start des Programms aus dem Editor von Spyder heraus.

1. $NAXIS1 = 5781$ / Axis length = Anzahl der Pixel/Wellenlängenpunkte
2. $CRVAL1 = 5037.399870393083$ = Start der Wellenlängenskala in Angström
3. $CDELT1 = 0.028791494561172293$ = 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

$$\text{Wellenlänge}[99] = CRVAL1 + (99 - CRPIX1 + 1) * CDELT1$$

Die Indexzählung beginnt in Python mit 0, also ist $\text{Wellenlänge}[0] = CRVAL1 + (0 - CRPIX1 + 1) * CDELT1 =$ die Startwellenlänge des Spektrums auf der kurzwelligen Seite.

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

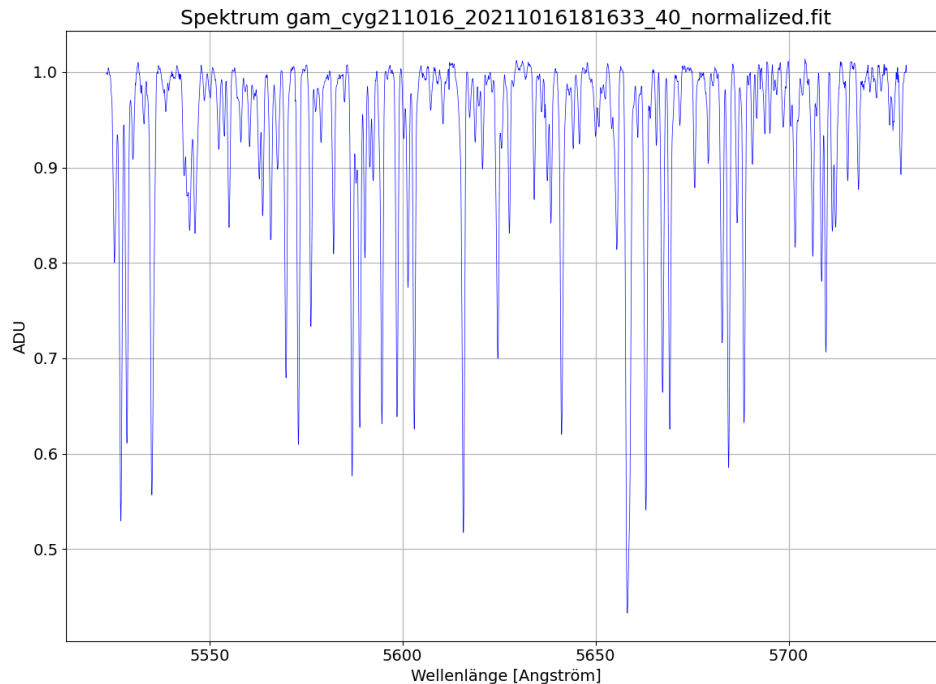


Figure 6.3: Durch das Skript 'spektrum_ansehen.py' erzeugter Plot eines Spektrums.

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.

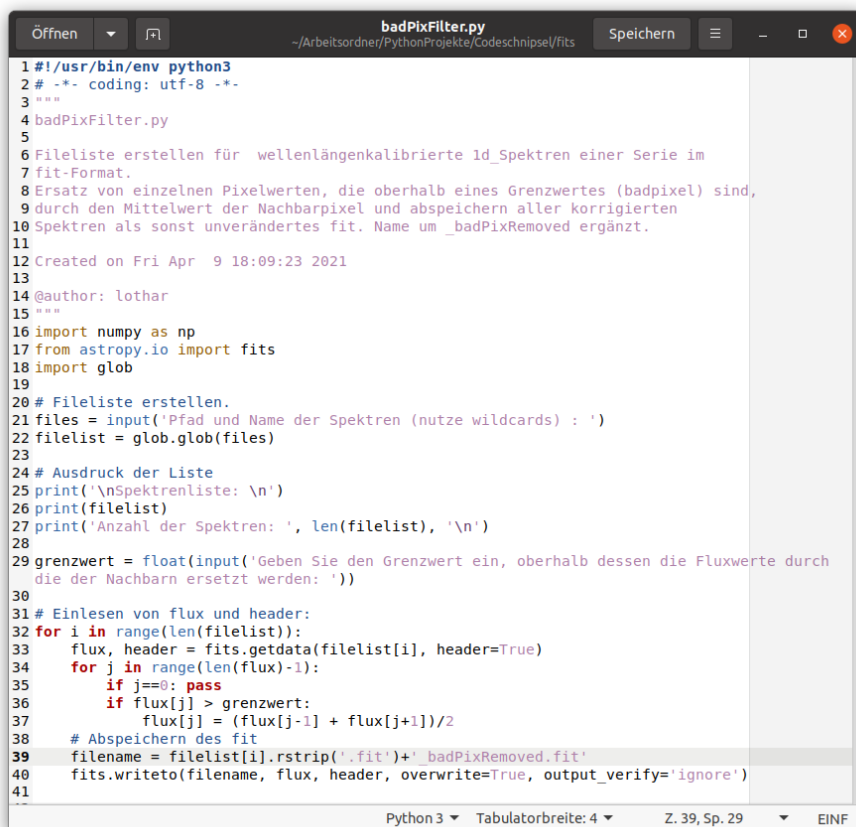
Part III.

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 Zigfaches 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
30die der Nachbarn ersetzt werden: '))
31
32# Einlesen von flux und header:
33for i in range(len(filelist)):
34    flux, header = fits.getdata(filelist[i], header=True)
35    for j in range(len(flux)-1):
36        if j==0: pass
37        if flux[j] > grenzwert:
38            flux[j] = (flux[j-1] + flux[j+1])/2
39    # Abspeichern des fit
40    filename = filelist[i].rstrip('.fit')+' _badPixRemoved.fit'
41    fits.writeto(filename, flux, header, overwrite=True, output_verify='ignore')
```

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

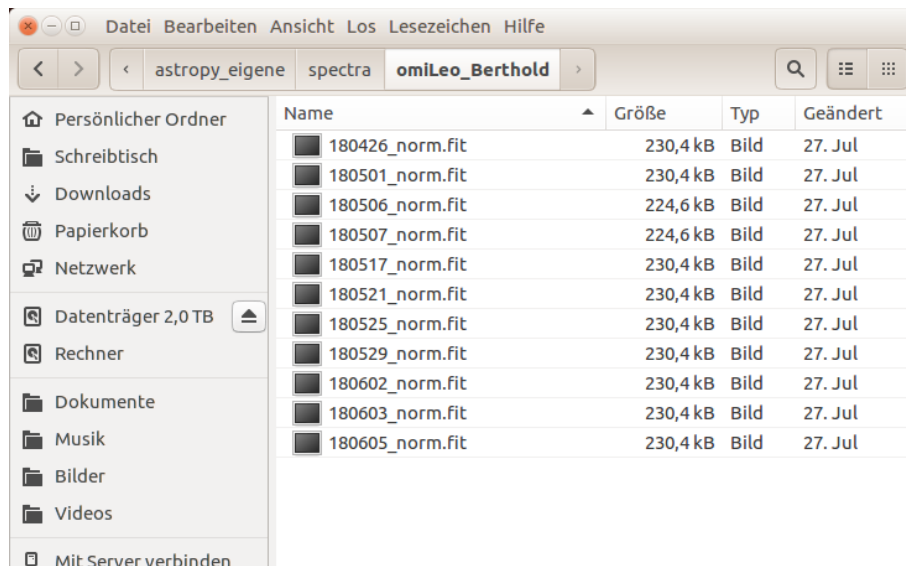


Figure 8.1: Zeitserie von normierten 1d-Echellespektren von omi Leo.

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.

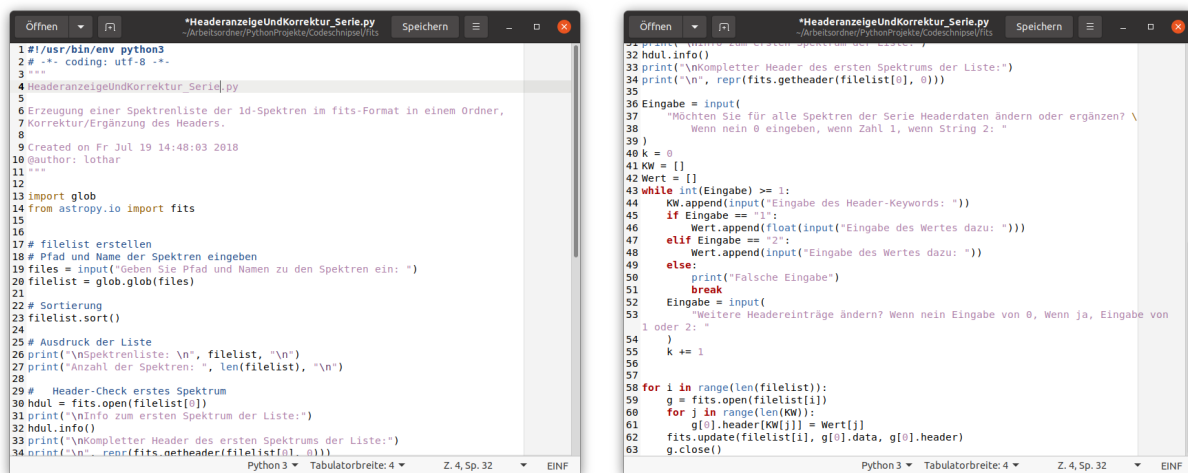


Figure 8.2: Programmlisting Header eines 1d-Spektrums (im fits-Format) anzeigen, ändern, aktualisieren.

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.

```

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  Anzeigen 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('Ausgabe 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
59
60
61
62
63
64
65
66
67
68 # Plot gesamtes Spektrum
69 plt.style.use('seaborn-white')
70 fig = plt.figure(1, figsize=(14, 10))
71 plt.plot(wave, flux)
72 plt.xlabel('Wellenlänge [Angström]', fontsize=15)
73 plt.xticks(fontsize=15)
74 plt.ylabel('ADU', fontsize=15)
75 plt.yticks(fontsize=15)
76 plt.title('Spektrum ' + file, fontsize=20)
77 plt.grid(True)
78 # fig.savefig(file.strip('.fit')+'.pdf')
79 # Kann aktiviert werden, wenn pdf der Grafik erwünscht.
80
81
82 # Erzeugen des Spektrumsausschnitts, plot und abspeichern als fit
83 print('Angabe des zu übernehmenden Wellenlängenbereichs:')
84 a = float(input('Anfang des Spektrums: '))
85 b = float(input('Ende des Spektrums: '))
86 aindex = int((a - crval) / cdel)
87 bindex = int((b - crval) / cdel)
88 filename = file.strip('.fit') + '_' + str(a) + '_' + str(b) + '.fit'
89 newflux = flux[aindex:bindex]
90 header['CRVAL1'] = crval + aindex * cdel
91 header['NAXIS1'] = bindex - aindex
92 header['CRPIX1'] = 1.
93 fits.writeto(filename, newflux, header, overwrite=True, output_verify='ignore')
94
95 # Abspeichern als ascii-File (.dat)
96 newwave = wave[aindex:bindex]
97 filename_dat = file.strip('.fit') + '_' + str(a) + '_' + str(b) + '.dat'
98 ascii.write([newwave, newflux], filename_dat, overwrite=True,
99            names=['WAVE', 'NFLUX'], format='tab')
100
101 # Plot
102 fig = plt.figure(2, figsize=(14, 10))
103 plt.plot(newwave, newflux)
104 plt.xlabel('Wellenlänge [Angström]', fontsize=18)
105 plt.xticks(fontsize=16)
106 plt.ylabel('ADU', fontsize=18)
107 plt.yticks(fontsize=16)
108 plt.title('Spektrum des Cep', fontsize=20)
109 plt.grid(True)
110 fig.savefig(filename.strip('.fit')+'.pdf')
111 # Kann durch Auskommentieren aktiviert werden, wenn pdf der Grafik erwünscht.

```

Figure 9.1: Programmlisting von „Crop_Ausschnitt_fits.py“.

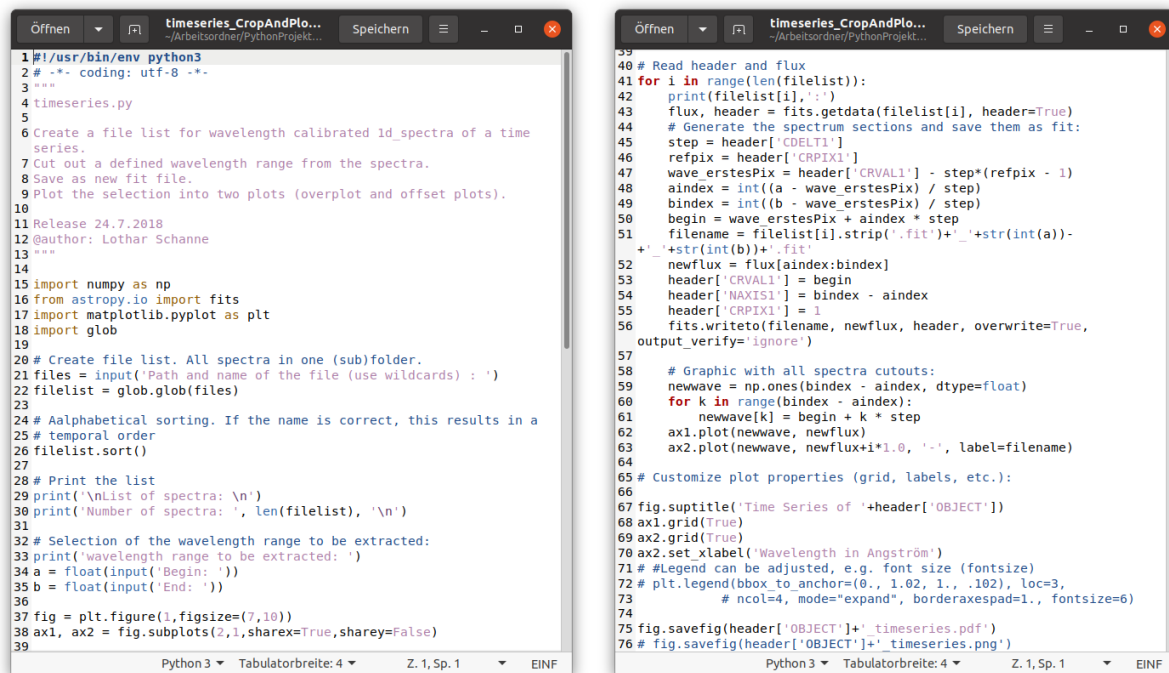


Figure 10.1: Skript zum Beschneiden und Plotten von Zeitreihen von 1d-Spektren (.fits).

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_Ausschnitt_Zeitreihe_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

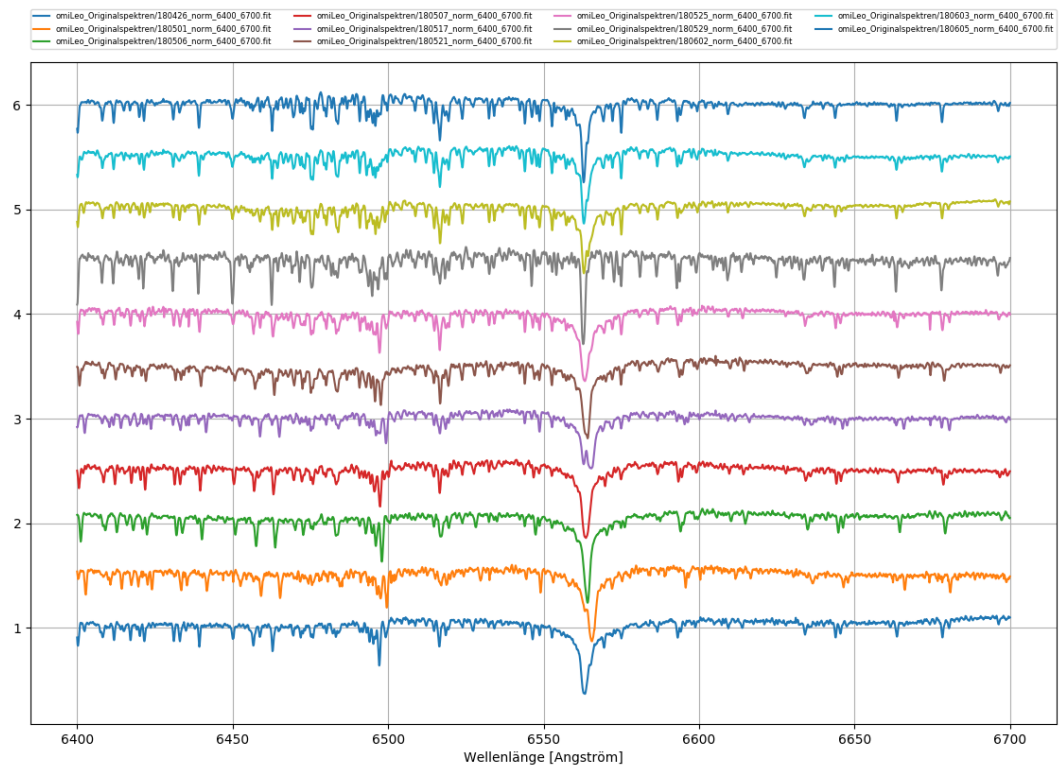
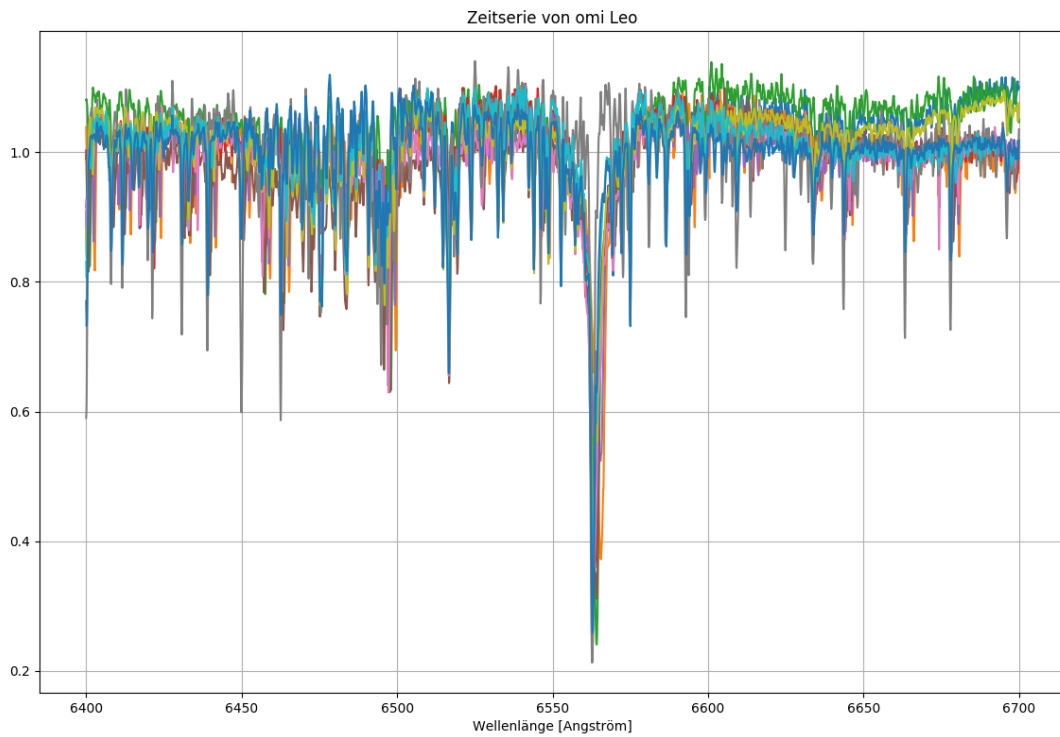


Figure 10.2: Plot einer Zeitreihe von 1d-Spektren. Oben alle übereinander geplottet, unten vertikal versetzt.



```
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_und_dat.py zur Umwandlung von 1d-Spektren im fits-Format in Ascii-Dateien.

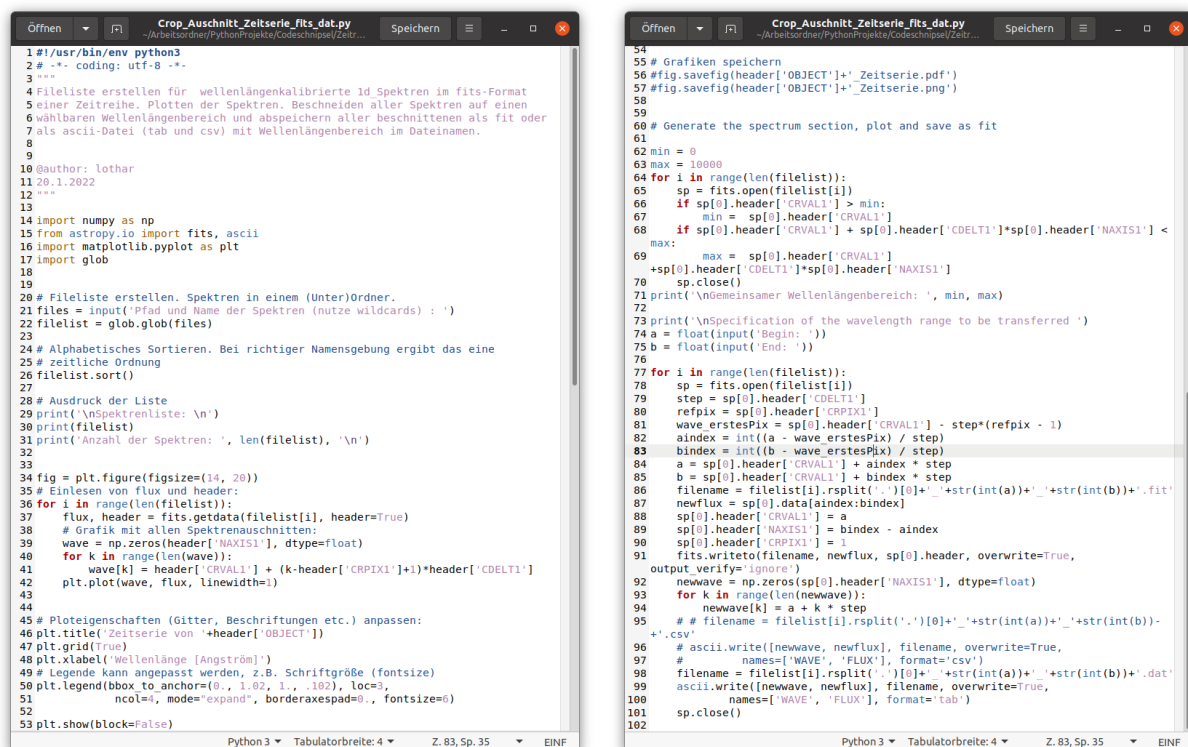


Figure 12.1: Code des Skripts Crop_Auschnitt_Zeitreihe_fits_dat.py zur Beschneidung einer Serie von 1d-Spektren im fits-Format und Abspeicherung der neuen Spektren als fits- und als ascii-Dateien.

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_mittleresSpektrum* (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#>

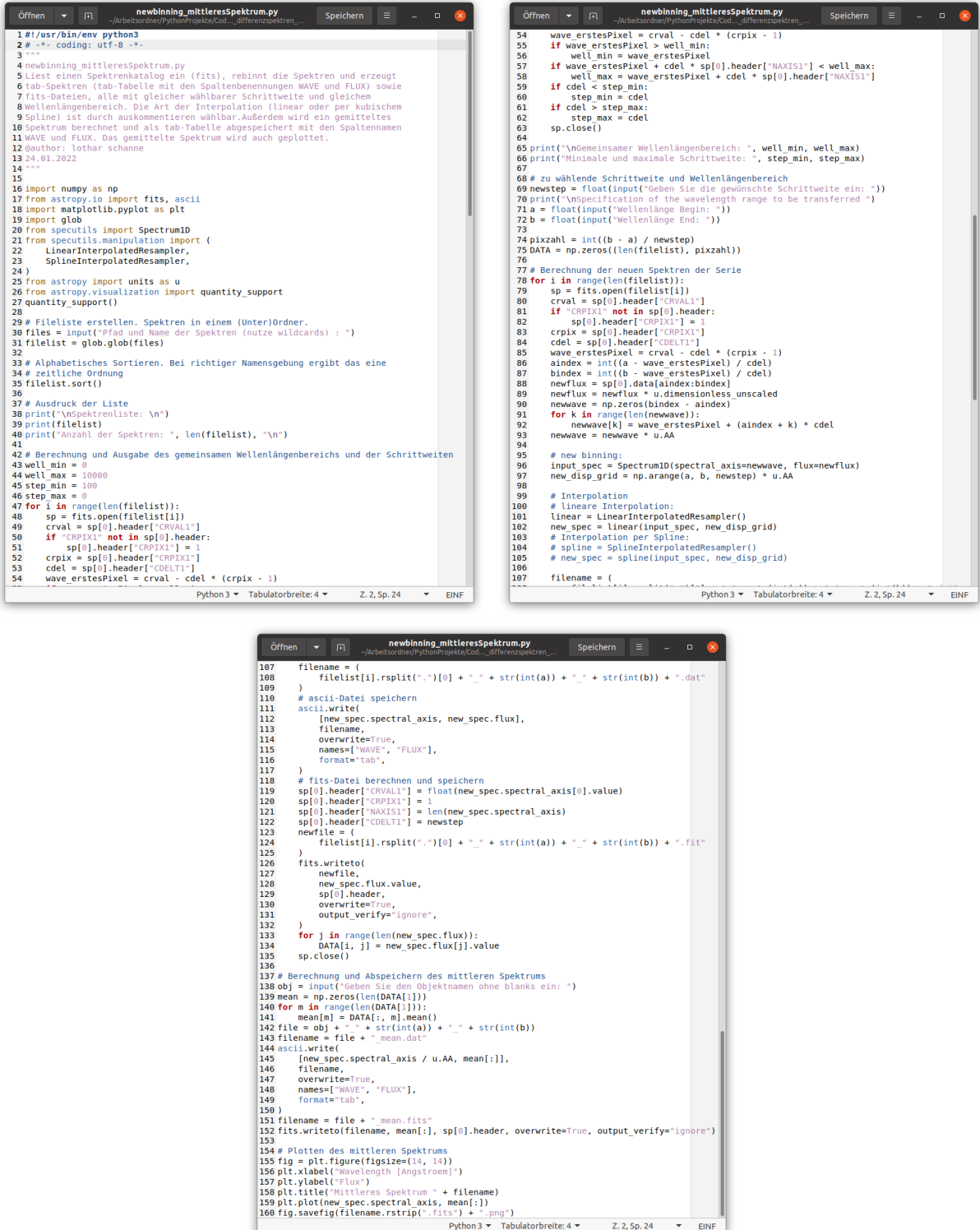


Figure 13.1: Skript zum Rebinnen einer Serie von 1d-Spektren im fits-Format auf einen gemeinsamen Wellenlängenbereich und gleiche Schrittweite sowie Mitteln der Spektren.

```

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe
Öffnen ▾ [Icon] 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)

```

Python ▾ Tabulatorbreite: 4 ▾ Z. 1, Sp. 1 ▾ EINF

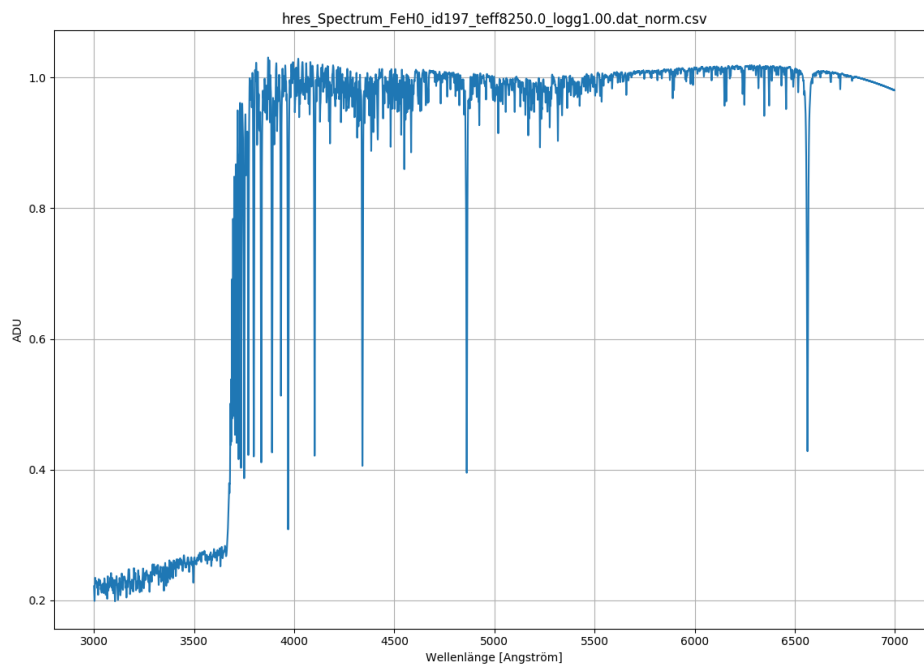


Figure 14.1: Listing von *1d_txt_plotten.py* und Abbildung eines ascii-1d-Spektrums erzeugt durch *1d_txt_plotten.py*.

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

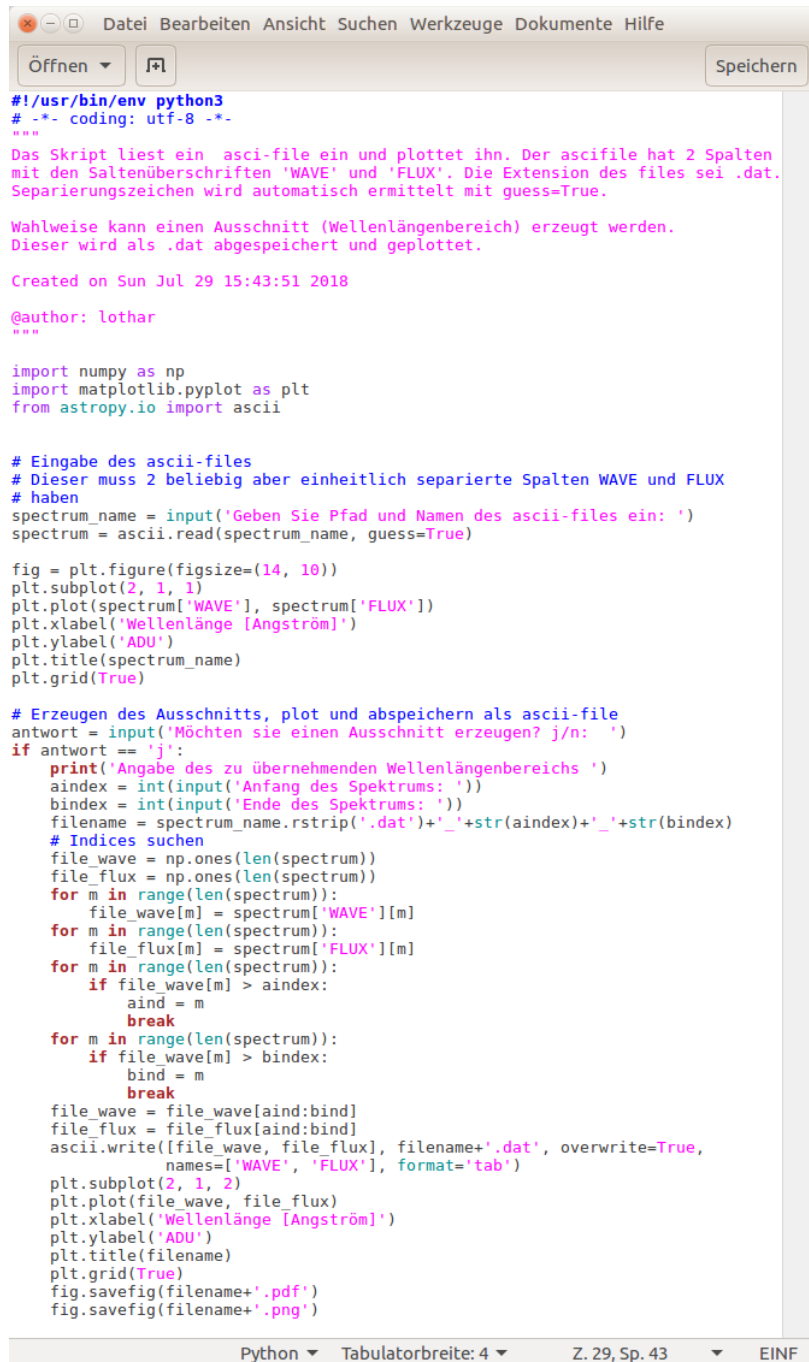
Manchmal möchte man ein hoch aufgelöstes 1d-Spektrum im fits-Format auf eine geringere Auflösung R bringen. Das kann mit dem Skript `convol_fits.py` geschehen. Nach dem Einlesen des Spektrums und der Eingabe des gewünschten R (z.B. 10000) wird das knvolvierte Spektrum berechnet und abgespeichert. Der 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.



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Das Skript liest ein ascii-file ein und plottet ihn. Der ascifile hat 2 Spalten
mit den Saltenüberschriften '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

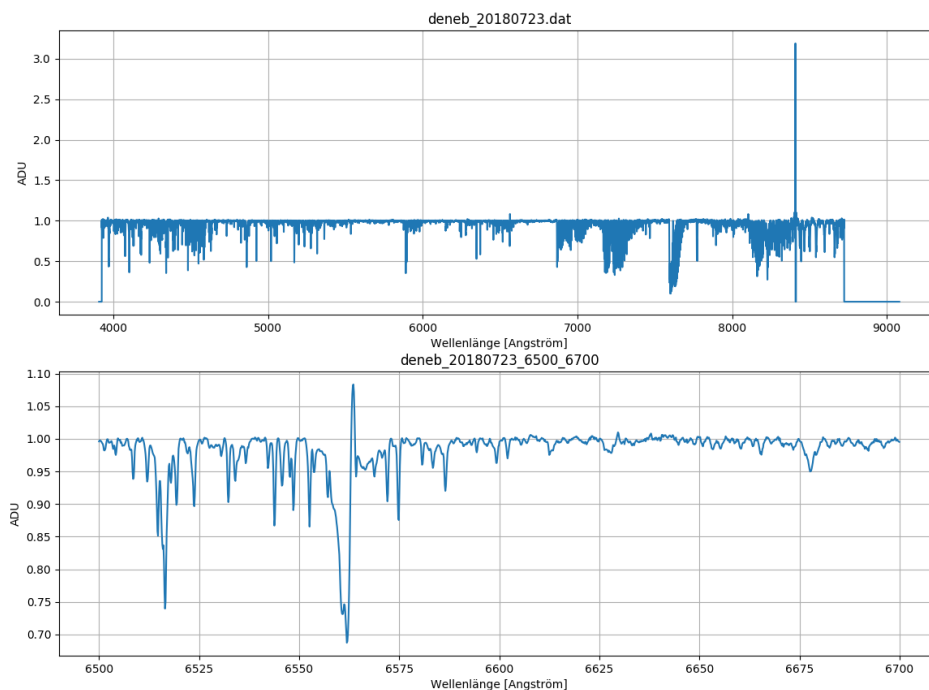


Figure 15.2: Resultierende Grafik des Skripts *PlotAndCrop_1d_ascii.py*. Oben das gesamte Echelle-Spektrum von Deneb, unten ein Ausschnitt von 6500 bis 6700 Angström.

Öffnen

Speichern

Python 3

Tabulatorbreite: 4

Z. 5, Sp. 1

EINF

```

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

```

Öffnen

Speichern

Python 3

Tabulatorbreite: 4

Z. 5, Sp. 1

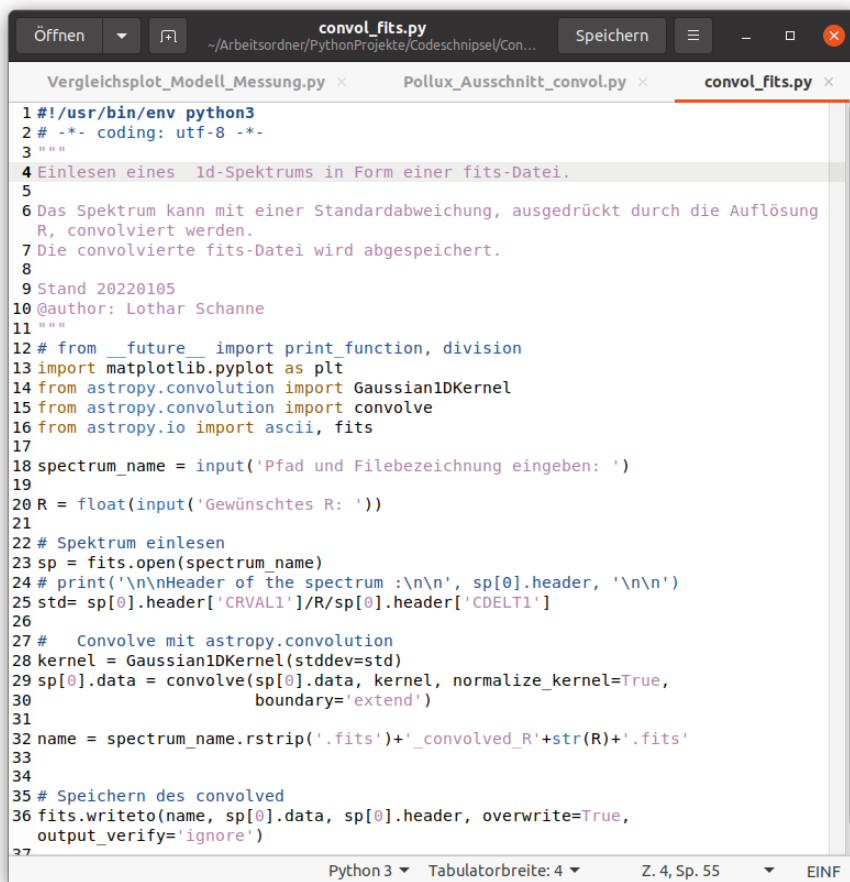
EINF

```

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("Wavelength [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)[0] + "_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['CDELT1']
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# Speichern des convolved
35fits.writeto(name, sp[0].data, sp[0].header, overwrite=True,
36            output_verify='ignore')
```

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

```
#!/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 zweispaltigen
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')
```

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_conv_spec.py* vorgestellt (Programmlisting in 19.1), welches Modellspektren aus der Pollux-Datenbank (<http://pollux.graal.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

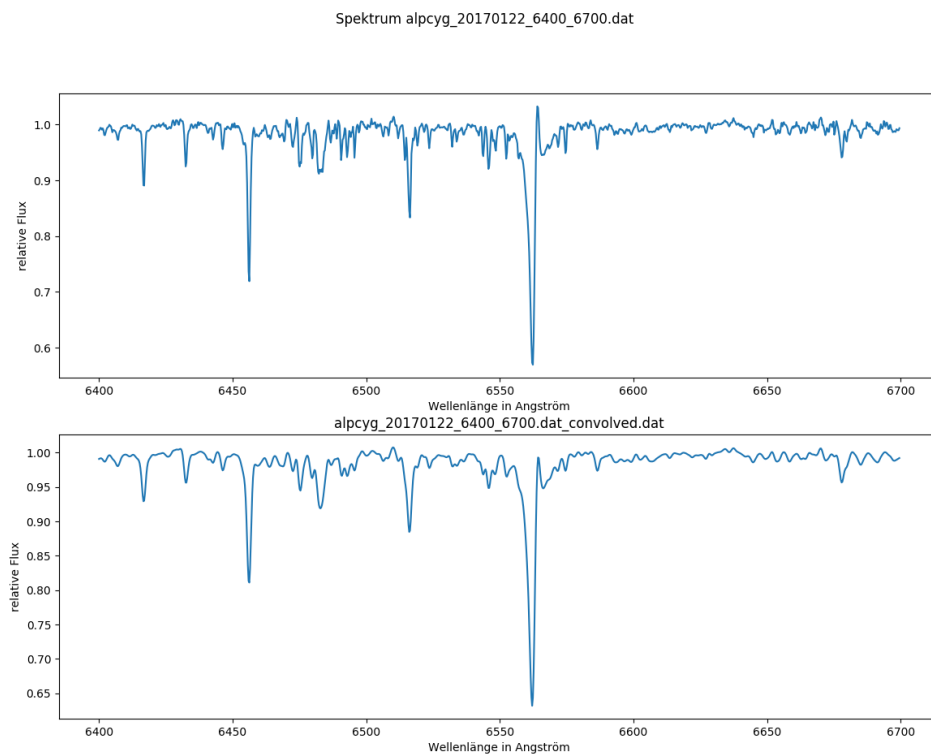


Figure 18.2: Ergebnisgrafik des Skripts *convol_dat.py* zum Falten eines Spektrums mit einer beliebigen Gaussfunktion. Hier wurde mit dem Parameter $std = 2$ (\approx Verdoppelung des gerade noch aufgelösten Elements, Halbierung der relativen Auflösung R) gearbeitet. Die Breite der Linien wurde etwa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


```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3'''
4Einlesen eines synthetischen Spektrums in Form einer Tabelle (wie als .spec
5in der Pollux-Datenbank erhältlich).
6Berechnung eines wählbaren Wellenlängenausschnitts (Pandas Dataframe mit
7'newtable' bezeichnet). Dieser Bereich wird dann mit einer wählbaren
8FWHM (Apparateprofil) gefaltet.
9Geplottet wird der gewählte rebinnte Wellenlängenbereich und zusätzlich das
10gefaltete Spektrum.
11Der rebinnte Flux-Ausschnitt des ursprünglichen .spec und der
12rebinnte und convolierte Flux wird zusammen mit der Wellenlänge in je einer
13zweispaltigen ascii-Tabelle (spacer = tab) und in einer fits-Datei
14abgespeichert.
15
16Stand 20220129
17@author: lothar
18'''
19# from __future__ import print_function, division
20import numpy as np
21import matplotlib.pyplot as plt
22from astropy.convolution import Gaussian1DKernel, convolve
23from astropy.io import ascii, fits
24from PyAstronomy.pyasl import binningx0dt
25import pandas as pd
26
27file = input("Pfad und Dateibezeichnung eingeben: ")
28table = pd.read_fwf(file, names=["WAVE", "AFUX", "NFLUX"], header=0)
29begin = float(
30    input("Geben Sie den Beginn des gewünschten Wellenlängenbereichs in
31    Angström ein: ")
32)
33end = float(input("Geben Sie das Ende des gewünschten Wellenlängenbereichs
34    ein: "))
35fwhm = float(
36    input("Geben Sie die gewünschte FWHM des Apparateprofils in Angström ein: ")
37)
38step = float(input("Geben Sie die gewünschte Schrittweite des
39    Ergebnisspektrums ein: "))
40
41newtable = table[table["WAVE"] >= begin]
42newtable = newtable[newtable["WAVE"] <= end]
43
44# Binning
45data, d = binningx0dt(
46    newtable["WAVE"],
47    newtable["NFLUX"],
48    dt=step,
49    x0=newtable["WAVE"].min(),
50    x0=newtable["WAVE"].min(),
51    removeEmpty=False,
52)
53Data = data.T
54wave = Data[0]
55flux = Data[1]
56
57# Convolve mit astropy.convolution
58kernel = Gaussian1DKernel(stddev=fwhm / 2.3 / step)
59convolved = convolve(flux, kernel, normalize_kernel=True, boundary="extend")
60
61# Grafik
62fig, ax = plt.subplots(2)
63plt.xlabel("Wellenlänge [Angström]", fontsize=5)
64fig.suptitle("Pollux-Spektrum " + file, fontsize=5)
65ax[0].plot(wave, flux, linewidth=0.1)
66ax[0].set_xlim(4000, 6700)
67ax[0].set_ylim(0.2, 1.1)
68ax[0].tick_params(axis="both", labelsize=3)
69ax[0].grid(b=True, axis="both", linewidth=0.2)
70ax[1].plot(wave, convolved, linewidth=0.2)
71ax[1].set_xlim(4000, 6700)
72ax[1].set_ylim(0.2, 1.1)
73ax[1].grid(b=True, axis="both", linewidth=0.2)
74name = file.rstrip(".spec")
75plt.savefig(file + ".png")
76plt.savefig(name + ".pdf")
77
78# Speichern des beschnittenen .spec, NFLUX jetzt als FLUX als dat und fits
79
80name = (
81    file.rstrip(".spec") + " " + str(int(begin)) + " " + str(int(end)) +
82    " _rebinned.dat"
83)
84
85name = (
86    file.rstrip(".spec")
87    + " "
88    + str(int(begin))
89    + " "
90    + str(int(end))
91    + " _rebinned.fits"
92)
93
94header = fits.Header()
95header["SIMPLE"] = "T"
96header["BITPIX"] = -32

```

```

95header["SIMPLE"] = "T"
96header["BITPIX"] = -32
97header["NAXIS"] = 1
98header["CRVAL1"] = wave[0]
99header["NAXIS1"] = len(wave)
100header["CDELT1"] = step
101header["CUNIT1"] = "Angstrom"
102header["CTYPE1"] = "Wavelength"
103header["CRPIX1"] = 1
104
105fits.writeto(
106    name, flux, header, overwrite=True, output_verify="ignore",
107)
108
109# convoliert
110name = (
111    file.rstrip(".spec")
112    + " "
113    + str(int(begin))
114    + " "
115    + str(int(end))
116    + " _rebinned_convolved.dat"
117)
118
119ascii.write(
120    [wave, convolved], name, overwrite=True, names=["WAVE", "FLUX"],
121    format="tab"
122)
123
124name = (
125    file.rstrip(".spec")
126    + " "
127    + str(int(begin))
128    + " "
129    + str(int(end))
130    + " _rebinned_convolved.fits"
131)
132
133header = fits.Header()
134header["SIMPLE"] = "T"
135header["BITPIX"] = -32
136header["NAXIS"] = 1
137header["CRVAL1"] = wave[0]
138header["NAXIS1"] = len(wave)
139header["CDELT1"] = step
140header["CUNIT1"] = "Angstrom"
141header["CTYPE1"] = "Wavelength"
142header["CRPIX1"] = 1
143
144fits.writeto(
145    name, convolved, header, overwrite=True, output_verify="ignore",
146)

```

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

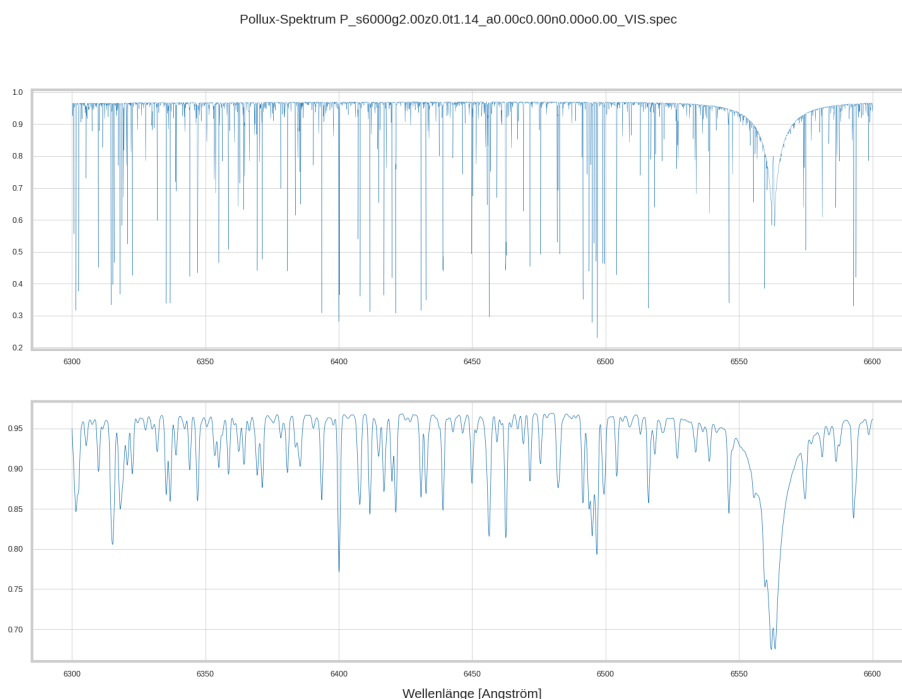


Figure 19.2: Grafik des Skripts *Pollux_Ausschnitt_rebinned_convolve_spec.py*, oben der rebinnte Wellenlängenausschnitt des gesamten Spektrums aus der Pollux-Datenbank, unten der zusätzlich mit dem Apparateprofil (hier 1 Angström) convolierte Ausschnitt.

Spektrumausschnitt und darunter der rebinnte und convolierte Ausschnitt. Abgespeichert werden die Dateien des rebinnten 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 Spektrographenauflösung.
6Auswahl des Wellenlängenbereiches und
7Anpassung auf eine gewünschte Schrittweite. Speicherung als ascii-tab-Tabelle
8und als .fit mit dem einzugebenden 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('Filebezeichnung für Speicherung eingeben: ')+'_'
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) + '_' + str(int(R))
39
40print('\nEingabe des gewünschten Wellenlängenbereichs 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, useMeanX=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_Gaussbroadenedes.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.

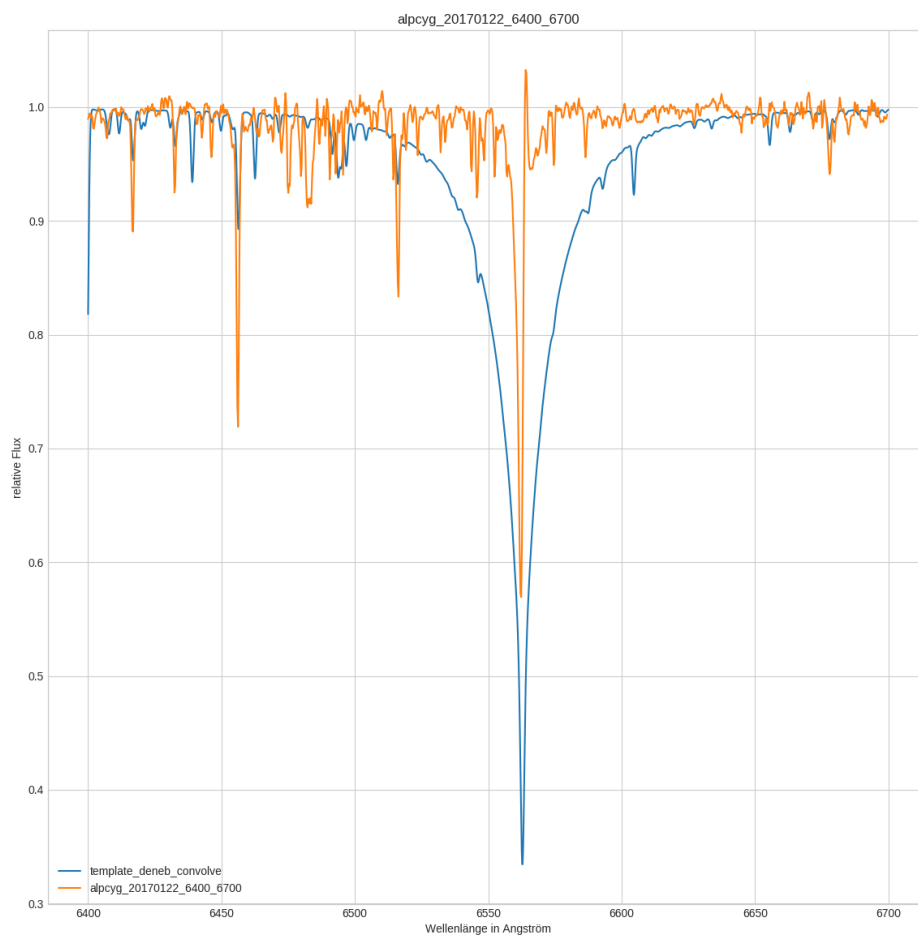


Figure 21.2: Von *Vergleichsplot_Modell_Messung.py* erzeugte Grafik.

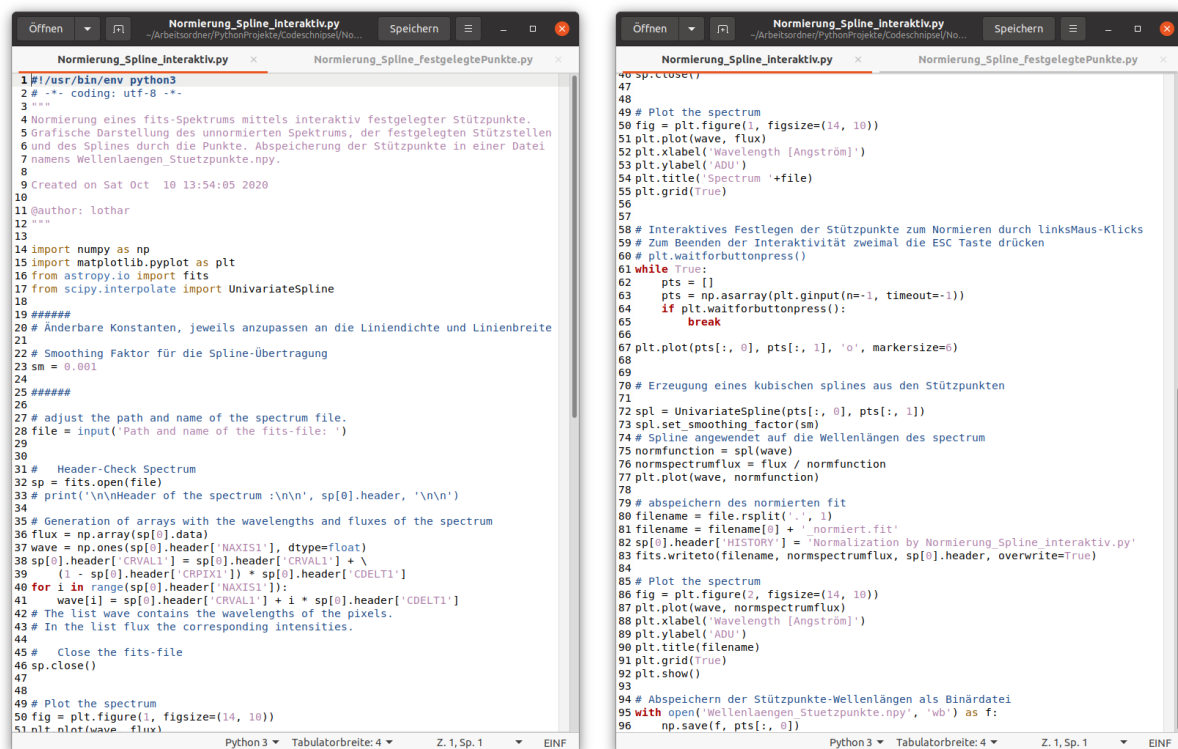


Figure 22.1: Pythonskript zur interaktiven Normierung eines 1d-Spektrums im fits-Format.

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

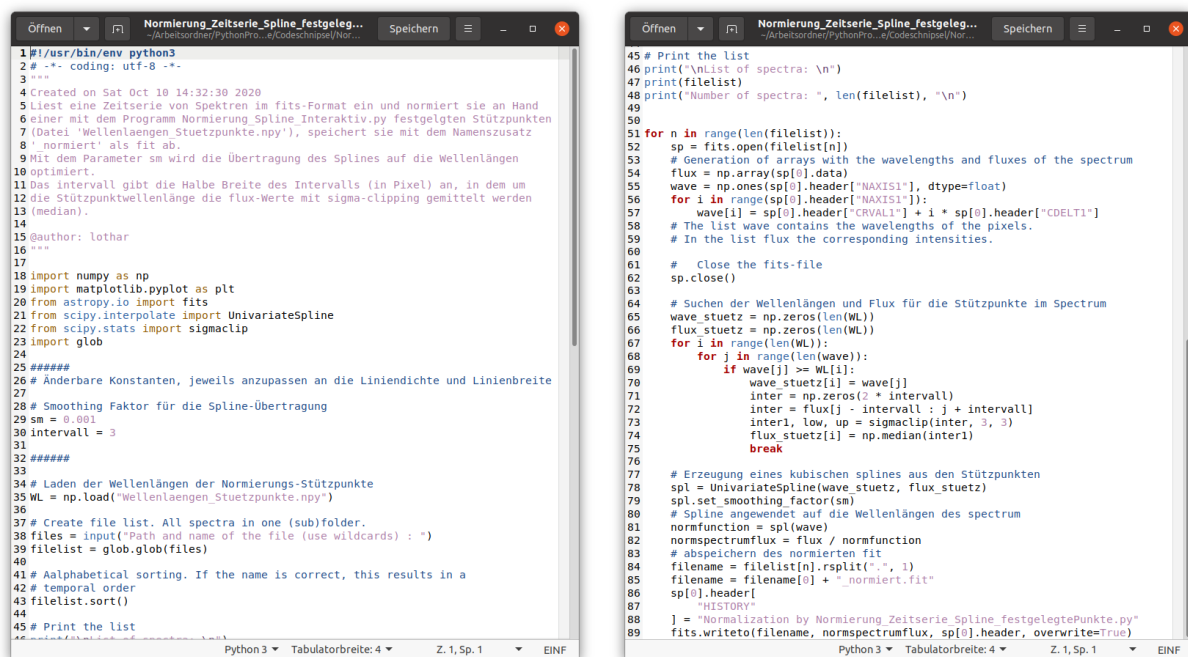


Figure 23.1: Skript zur Normierung einer Serie von 1d-Spektren im fits-Format.

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 \AA), 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"""
5Works for a single or a time series of 1d spectra in fits format.
6At first spectrum the parameters 'inter' and 'sm' are optimized.
7Takes from pixel to pixel an interval of 'inter' pixels, calculated
8of which is a percentile and compares it with the neighboring intervals.
9Finds so local maxima = support points.
10You can also distinguish several wavelength ranges from the formation of
11normalization points, i.e. delete them (wide lines).
12The Smoothing parameter 'sm' adjusts the sensitivity of the spline
13which is used for normalization at the end. After that, you can still delete
14points that are a certain percentage larger or smaller than the spline.
15Finally, all spectra of the time series are normalized with the optimized
16parameters.
17
1807.12.2020
19@author: lothar
20"""
21
22import numpy as np
23import matplotlib.pyplot as plt
24from astropy.io import fits
25from scipy.stats import sigmaclip
26import glob
27import concurrent.futures as futures
28import os
29from csaps import csaps
30
31# from numba import jit
32
33
34# definition of functions
35
36
37def 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["CDELT1"]
44    for i in np.arange(header["NAXIS1"]):
45        wave[i] = header["CRVAL1"] + i * header["CDELT1"]
46    wave = wave[flux >= flux_median * 0.001]
47    flux = flux[flux >= flux_median * 0.001]
48    return wave, flux
49
50def maxmacalc(inter, flux, wave):
51    pixelanzahl = len(wave)
52    maxmas_flux = np.zeros(pixelanzahl)
53    maxmas_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            maxmas_flux[n + inter // 2] = np.percentile(interflux1, 50)
64            maxmas_wave[n + inter // 2] = wave[n + inter // 2]
65    points_flux = maxmas_flux[maxmas_flux != 0]
66    points_wave = maxmas_wave[maxmas_wave != 0]
67    return points_flux, points_wave
68
69
70# @jit
71def 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# @jit
84def cutting(wave, points_wave):
85    for n in np.arange(len(wave)):
86        if wave[n] < points_wave[0]:
87            wave[n] = 0
88        if wave[n] > points_wave[-1]:
89            wave[n] = 0
90    return wave
91
92def pipe(kl):
93    flux, header = fits.getdata(filelist[kl], header=True)
94    # Generation of arrays with the wavelengths and fluxes of the spectrum
95    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 = maxmacalc(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        + " Number of points "
139        + str(len(points_wave))
140    )
141    print(druck)
142
143
144    return (
145        kl,
146        normfunction,
147        normspectrumflux,
148        points_flux,
149        points_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
158inter = 10 # Intervall (Pixel)
159
160sm = 0.001 # initial smoothing factor for spline
161
162#####
163
164# plt.ion()
165
166# *****
167# Create file list. Spectra in a (sub)folder.
168files = input("Path and name of the spectra (use wildcards) : ")
169filelist = glob.glob(files)
170
171# Sort alphabetically. If the spectrum files are named correctly, this results
172# in a temporal order.
173filelist.sort()
174
175# Printout of the list for control purposes.
176print("List of spectra:")
177print(filelist)
178print("Number of spectra: ", len(filelist), "\n")
179
180# ***** Optimizing the parameters on the first spectrum
181print(
182    "We first optimize the parameter 'interval width' and select wavelength
183    ranges in which no interpolation points are desired.\n"
184)
185k = 0
186flux, header = fits.getdata(filelist[0], header=True)
187print("\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.

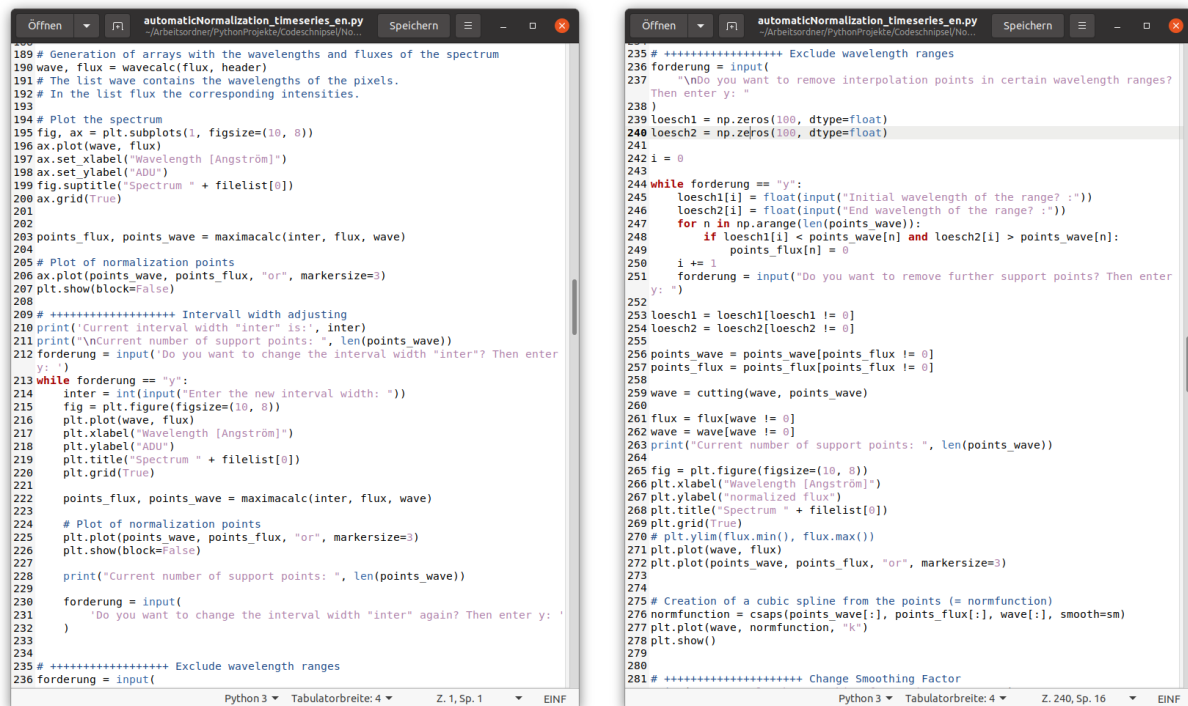


Figure 24.3: Pythonskript zum automatischen Normieren von Spektren im fits-Format. Fortsetzung.

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     "With the Smoothing factor we influence the 'detail' of the spline: \nsm =
285     1 means that the spline goes through all points, \nsm = 0 means linear
286     regression through all points (straight line). \nSelect a number between 0. and
287     1."
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
302     enter y: ")
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
312         the spline, enter y: "
313     )
314 )
315 while frage2[j] == "y":
316     s1.append(float(input("Enter s1 in %: ")))
317     s2.append(float(input("Enter s2 in %: ")))
318     print("Please wait")
319     points_flux = reduction(s1[j], s2[j], points_wave, points_flux, wave,
320                             normfunction)
321     points_wave = points_wave[points_flux != 0]
322     points_flux = points_flux[points_flux != 0]
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
345             and s2 % above the spline, enter y: "
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 Stützstellenanzahl 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 frase3 == "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
394 # saving the normfunction
395 antwort1 = input("If you want to save the normfunction as fit, then enter y: ")
396 if antwort1 == "y":
397     header["CRVAL1"] = wave[0]
398     header["NAXIS1"] = len(wave)
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 frase1 = 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 frase4 = 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 )

```

```

414 normalization points and the normalization function? Then enter y: "
415 frase5 = 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 frase1 == "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                     ) = f.result()
439                     del fs[f]
440                     # Saving the normalized spectrum as fit
441                     if antwort == "y":
442                         filename = filelist[k].rsplit(".", 1)
443                         filename = filename[0] + "_normalized.fit"
444                         header["CRVAL1"] = wave[0]
445                         header["NAXIS1"] = len(wave)
446                         header["CRPIX1"] = 1
447                         fits.writeto(
448                             filename,
449                             normspectrumflux,
450                             header,
451                             overwrite=True,
452                             output_verify="ignore",
453                         )
454                     # saving the normfunction
455                     if antwort1 == "y":
456                         header["CRVAL1"] = wave[0]
457                         header["NAXIS1"] = len(wave)
458

```

```

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 frase4 == "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 frase3 == "y":
478             filename = filelist[k].rsplit(".", 1)
479             filename = filename[0] + "_points.pdf"
480             plt.savefig(filename)
481             plt.show()
482
483     if frase5 == "y":
484         # Plot the normalized spectrum
485         fig = plt.figure(figsize=(10, 8))
486         plt.plot(wave, normspectrumflux)
487         plt.xlabel("Wavelength [Angström]")
488         plt.ylabel("normalized flux")
489         plt.title("Spectrum " + filelist[k])
490         plt.grid(True)
491         if frase3 == "y":
492             filename = filelist[k].rsplit(".", 1)
493             filename = filename[0] + "_normalized.pdf"
494             plt.savefig(filename)
495             plt.show()
496
497     fertig = len(res.not_done) == 0
498
499 x = input("Type e when you are finished viewing the graphics: ")
500 while x:
501     if x == "e":
502         print("END of program, all done")
503         x = False
504     else:
505         x = input(
506             "The input was not e.Type e when you are finished viewing the
graphics: "
507         )
508

```

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

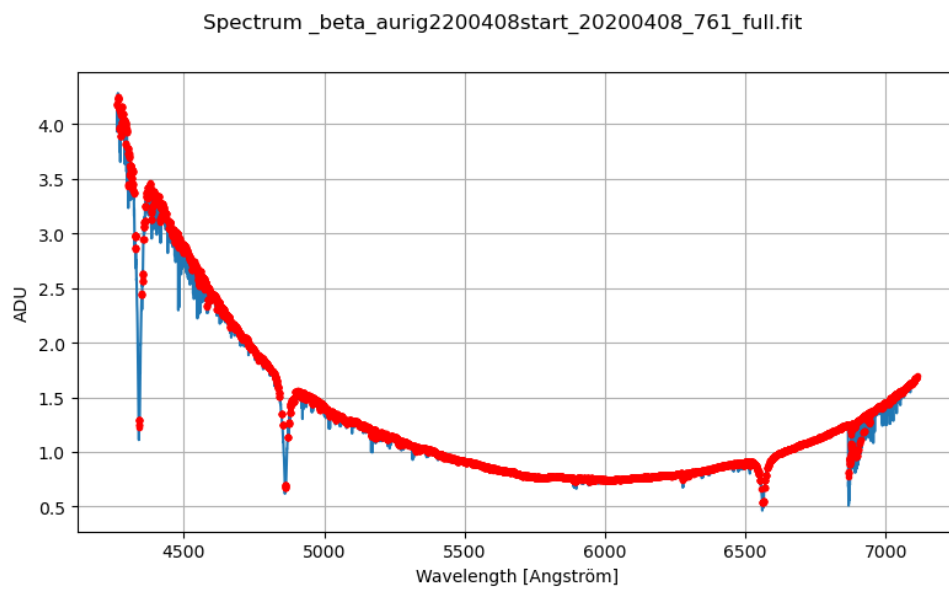


Figure 24.6: Grafik nach Start des Skripts.

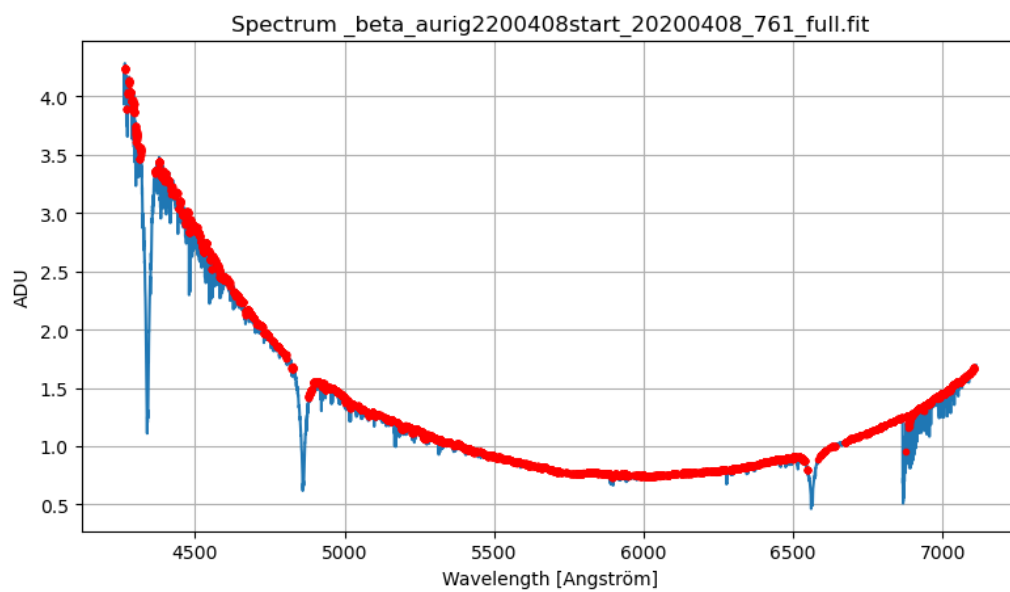


Figure 24.7: Grafik nach Änderung der Intervallbreite von 10 auf 30 Pixel.

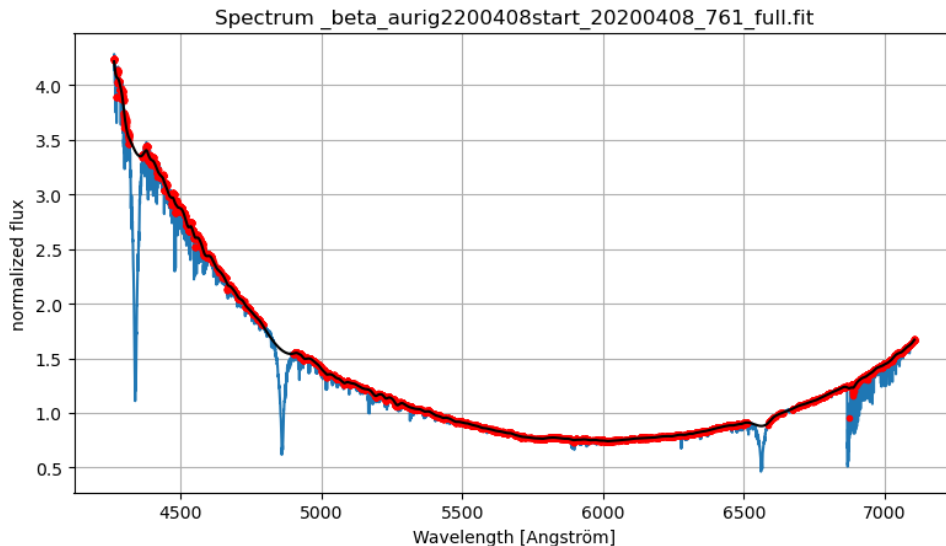


Figure 24.8: Grafik nach Ausschluß der Wellenlängenbereiche um die H α - und der H β -Linie. Der berechnete Spline ist als schwarze Kurve eingezeichnet.

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.

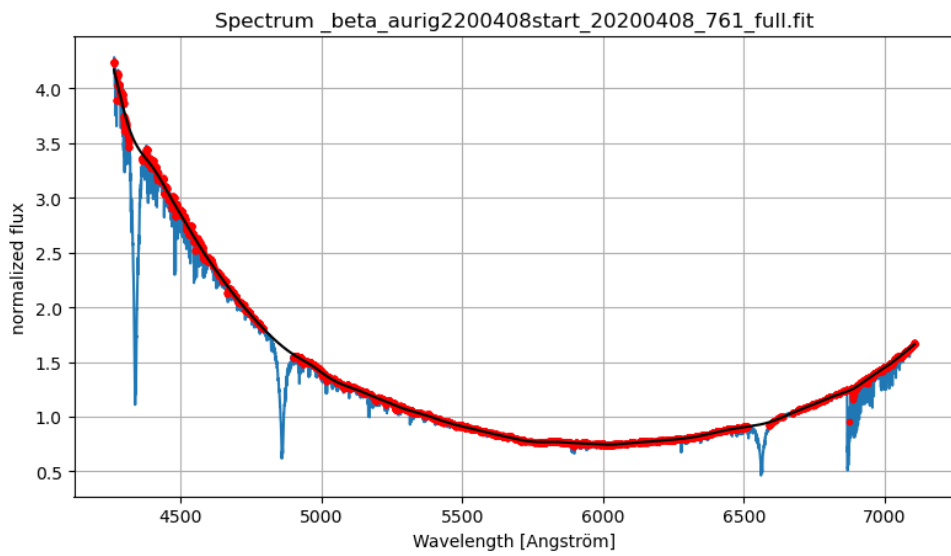


Figure 24.9: Spline nach Anpassung des Smoothingfaktors sm auf 0.000001.

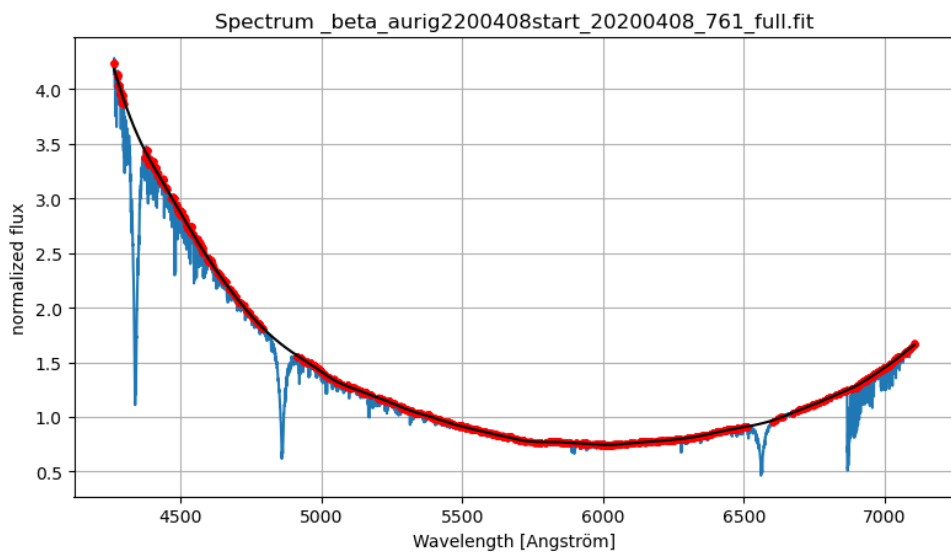


Figure 24.10: Grafik nach Löschen von Stützstellen außerhalb des Bands -0.5% bis $+5\%$ des Splines.

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

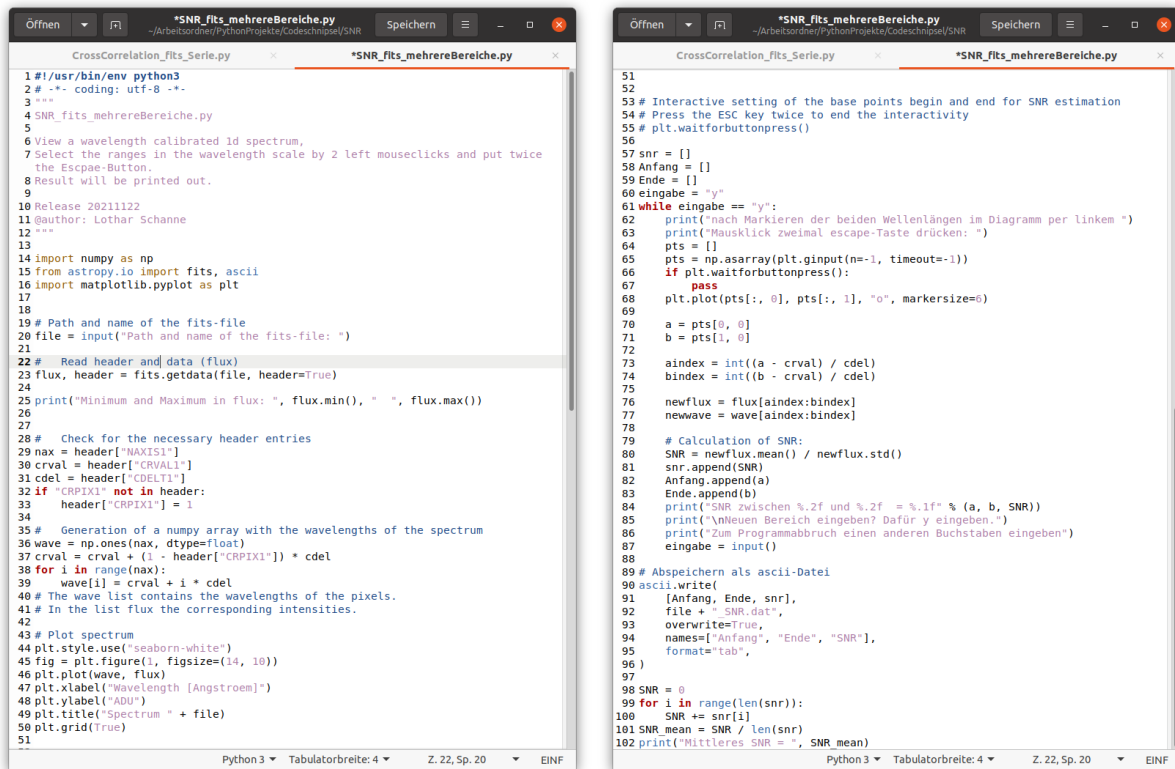


Figure 24.11: Skript zur Bestimmung des SNR in mehreren Wellenlängenbereichen eines 1d-Spektrums.

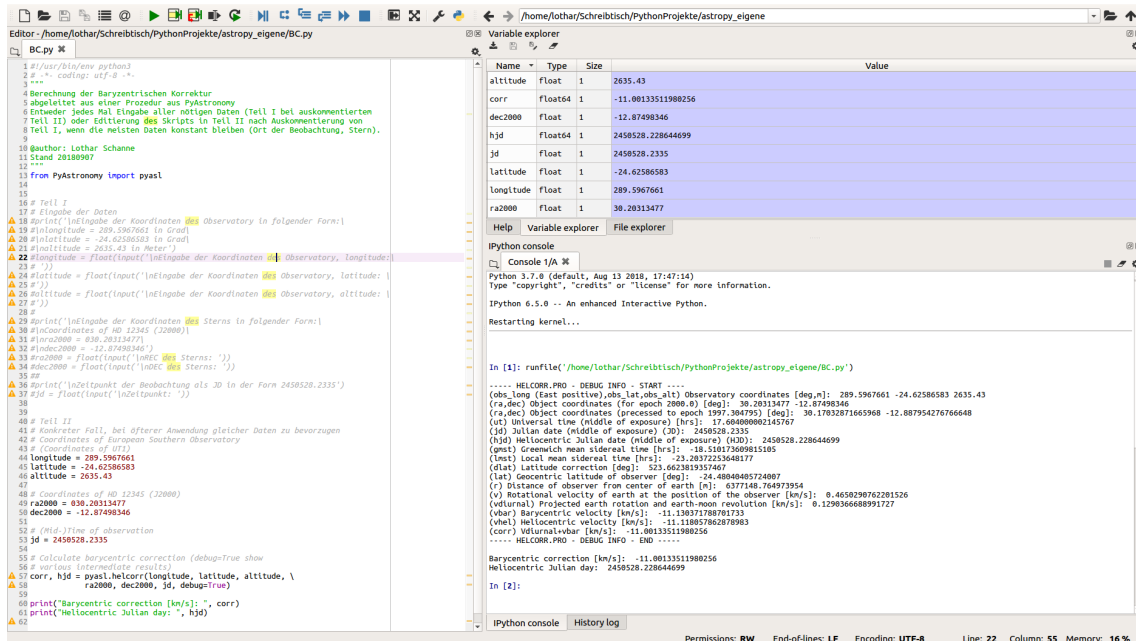


Figure 25.1: Berechnung der Baryzentrischen Korrektur mit dem Skript BC.py.

(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.

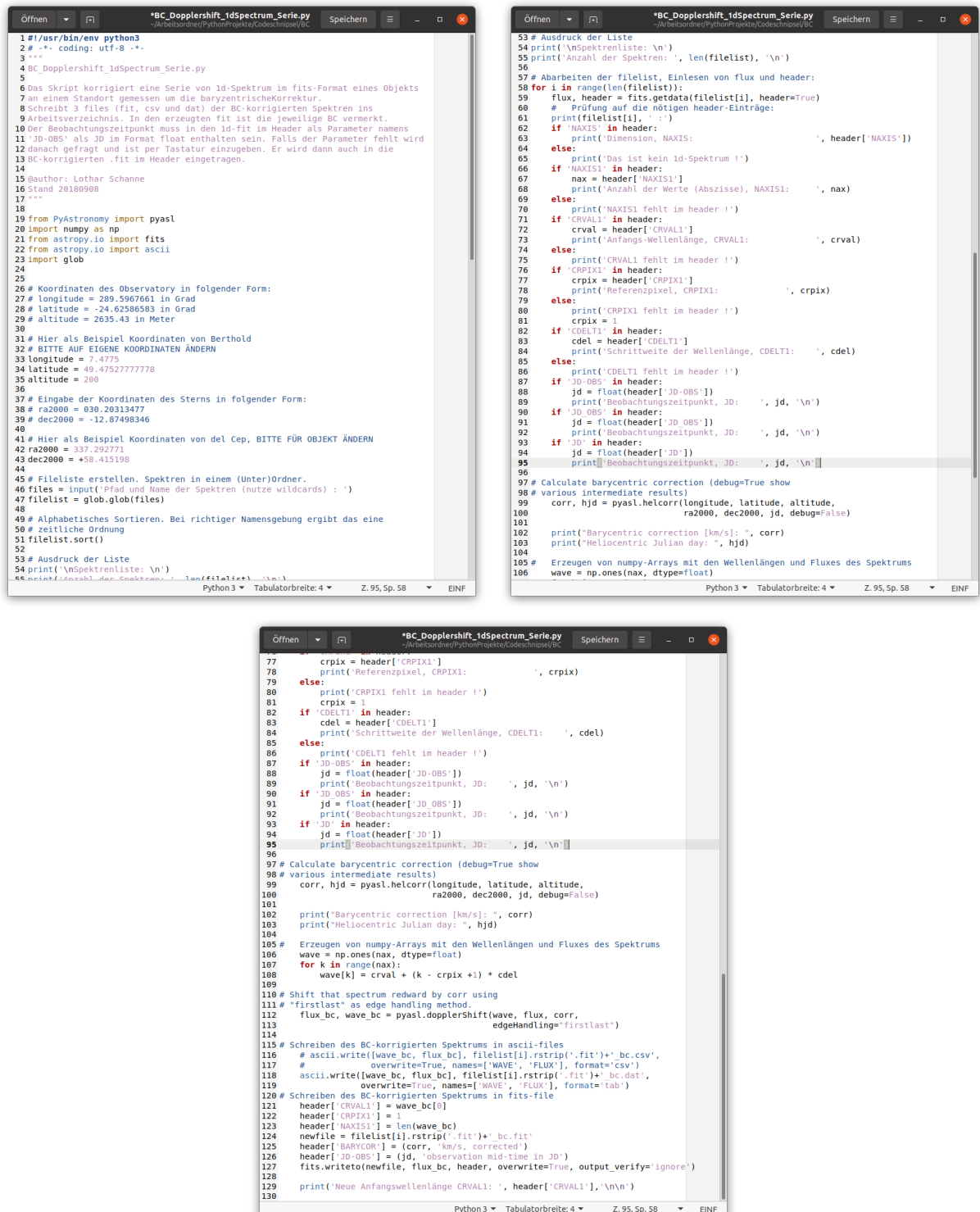


Figure 26.1: Programmlisting von `BC_Dopplershift_1dSpectrum_Serie.py`.


```

173 miniwave1 = np.zeros(len(filelist))
180 miniwave2 = 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") + "_" + lineie)
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") + "_" + lineie + ".png")
225
226 # Abspeichern als ascii-Datei
227 ascii.write(
228     [filelist, RV1, apex1, RV2, apex2],
229     lineie + "_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 Grafikfenster mit dem Spektrumausschnitt und den angeklickten Minima als png-Bild gespeichert. Nach der Bearbeitung aller Spektren der Serie werden die RV's und Apex 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 Mausklick 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 Grafikfenster 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 miniwave1 = np.zeros(len(filelist))
179 miniwave2 = np.zeros(len(filelist))
180 miniwave1 = np.zeros(len(filelist))
181 miniwave2 = np.zeros(len(filelist))
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     intervall = np.extract(abs(ts.columns[0] - wellenlaenge) < suchintervall, ts)
191     a = np.zeros(len(intervall))
192     b = np.zeros(len(intervall))
193     for j in range(len(intervall)):
194         a[j], b[j] = intervall[j]
195
196     fig = plt.figure(i)
197     plt.plot(a, b, "-", linewidth=0.5)
198     plt.title(filelist[i].rstrip(".dat") + "_" + linie)
199
200     print("\nSpektrum ", filelist[i])
201     print("Klicke das Minimum der ersten Linie an:")
202     pts = np.asarray(plt.ginput(n=1, timeout=-1))
203
204     # Das folgende Intervall [Angström] anpassen, bei verrauschten Linien breit,
205     # bei nicht verrauschten, schmalen Linien klein.
206     linienminimum_wave1 = a[abs(a - pts[0, 0]) <= 0.5]
207     linienminimum_flux1 = b[abs(a - pts[0, 0]) <= 0.5]
208
209     # Regression, Grad anpassen (2 oder 4 oder 6)
210     grad = 4
211     model = np.poly1d(np.polyfit(linienminimum_wave1, linienminimum_flux1, grad))
212
213     polyline = np.linspace(linienminimum_wave1[0], linienminimum_wave1[-1], 100)
214     modflux = model(polyline)
215
216     # Linienminimum berechnen
217     miniwave1[i] = modflux.min()
218     miniwave1[i] = polywave(modflux.argmin())
219
220     # Plotten
221     plt.plot(polyline, modflux, "--")
222     plt.plot(miniwave1[i], miniwave1[i], "o", color="black")
223     plt.show()
224     plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")
225
226 miniwave1[i] = polywave(modflux.argmin())
227
228     # Plotten
229     plt.plot(polyline, modflux, "--")
230     plt.plot(miniwave1[i], miniwave1[i], "o", color="black")
231     plt.show()
232     plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")
233
234     # Das folgende Intervall [Angström] anpassen, bei verrauschten Linien breit,
235     # bei nicht verrauschten, schmalen Linien klein.
236     linienminimum_wave2 = a[abs(a - pts[1, 0]) <= 0.3]
237     linienminimum_flux2 = b[abs(a - pts[1, 0]) <= 0.3]
238
239     # Regression
240     model = np.poly1d(np.polyfit(linienminimum_wave2, linienminimum_flux2, grad))
241
242     polyline = np.linspace(linienminimum_wave2[0], linienminimum_wave2[-1], 100)
243     modflux = model(polyline)
244
245     # Linienminimum berechnen
246     miniwave2[i] = modflux.min()
247     miniwave2[i] = polywave(modflux.argmin())
248
249     # Plotten
250     plt.plot(polyline, modflux, "--")
251     plt.plot(miniwave2[i], miniwave2[i], "o", color="black")
252
253     RV2[i] = (miniwave2[i] - wellenlaenge) / wellenlaenge * 299792
254     else:
255         RV2[i] = np.NaN
256
257     plt.savefig(filelist[i].rstrip(".dat") + "_" + linie + ".png")
258     plt.close(i - 1)
259
260 # Abspeichern als ascii-Datei
261 # *****
262 # *****
263 # *****
264 # *****
265 # *****
266 # *****
267 # *****
268 # *****
269 # *****
270 # *****
271 # *****
272 # *****
273 # *****
274 # *****
275 # *****
276 # *****
277 # *****
278 # *****
279 # *****
280 # *****
281 # *****
282 # *****
283 # *****
284 # *****
285 # *****
286 # *****
287 # *****
288 # *****
289 # *****
290 # *****
291 # *****
292 # *****
293 # *****
294 # *****
295 # *****
296 # *****
297 # *****
298 # *****
299 # *****
300 # *****
301 # *****
302 # *****
303 # *****
304 # *****
305 # *****
306 # *****
307 # *****
308 # *****
309 # *****
310 # *****
311 # *****
312 # *****
313 # *****
314 # *****
315 # *****
316 # *****
317 # *****
318 # *****
319 # *****
320 # *****
321 # *****
322 # *****
323 # *****
324 # *****
325 # *****
326 # *****
327 # *****
328 # *****
329 # *****
330 # *****
331 # *****
332 # *****
333 # *****
334 # *****
335 # *****
336 # *****
337 # *****
338 # *****
339 # *****
340 # *****
341 # *****
342 # *****
343 # *****
344 # *****
345 # *****
346 # *****
347 # *****
348 # *****
349 # *****
350 # *****
351 # *****
352 # *****
353 # *****
354 # *****
355 # *****
356 # *****
357 # *****
358 # *****
359 # *****
360 # *****
361 # *****
362 # *****
363 # *****
364 # *****
365 # *****
366 # *****
367 # *****
368 # *****
369 # *****
370 # *****
371 # *****
372 # *****
373 # *****
374 # *****
375 # *****
376 # *****
377 # *****
378 # *****
379 # *****
380 # *****
381 # *****
382 # *****
383 # *****
384 # *****
385 # *****
386 # *****
387 # *****
388 # *****
389 # *****
390 # *****
391 # *****
392 # *****
393 # *****
394 # *****
395 # *****
396 # *****
397 # *****
398 # *****
399 # *****
400 # *****
401 # *****
402 # *****
403 # *****
404 # *****
405 # *****
406 # *****
407 # *****
408 # *****
409 # *****
410 # *****
411 # *****
412 # *****
413 # *****
414 # *****
415 # *****
416 # *****
417 # *****
418 # *****
419 # *****
420 # *****
421 # *****
422 # *****
423 # *****
424 # *****
425 # *****
426 # *****
427 # *****
428 # *****
429 # *****
430 # *****
431 # *****
432 # *****
433 # *****
434 # *****
435 # *****
436 # *****
437 # *****
438 # *****
439 # *****
440 # *****
441 # *****
442 # *****
443 # *****
444 # *****
445 # *****
446 # *****
447 # *****
448 # *****
449 # *****
450 # *****
451 # *****
452 # *****
453 # *****
454 # *****
455 # *****
456 # *****
457 # *****
458 # *****
459 # *****
460 # *****
461 # *****
462 # *****
463 # *****
464 # *****
465 # *****
466 # *****
467 # *****
468 # *****
469 # *****
470 # *****
471 # *****
472 # *****
473 # *****
474 # *****
475 # *****
476 # *****
477 # *****
478 # *****
479 # *****
480 # *****
481 # *****
482 # *****
483 # *****
484 # *****
485 # *****
486 # *****
487 # *****
488 # *****
489 # *****
490 # *****
491 # *****
492 # *****
493 # *****
494 # *****
495 # *****
496 # *****
497 # *****
498 # *****
499 # *****
500 # *****
501 # *****
502 # *****
503 # *****
504 # *****
505 # *****
506 # *****
507 # *****
508 # *****
509 # *****
510 # *****
511 # *****
512 # *****
513 # *****
514 # *****
515 # *****
516 # *****
517 # *****
518 # *****
519 # *****
520 # *****
521 # *****
522 # *****
523 # *****
524 # *****
525 # *****
526 # *****
527 # *****
528 # *****
529 # *****
530 # *****
531 # *****
532 # *****
533 # *****
534 # *****
535 # *****
536 # *****
537 # *****
538 # *****
539 # *****
540 # *****
541 # *****
542 # *****
543 # *****
544 # *****
545 # *****
546 # *****
547 # *****
548 # *****
549 # *****
550 # *****
551 # *****
552 # *****
553 # *****
554 # *****
555 # *****
556 # *****
557 # *****
558 # *****
559 # *****
560 # *****
561 # *****
562 # *****
563 # *****
564 # *****
565 # *****
566 # *****
567 # *****
568 # *****
569 # *****
570 # *****
571 # *****
572 # *****
573 # *****
574 # *****
575 # *****
576 # *****
577 # *****
578 # *****
579 # *****
580 # *****
581 # *****
582 # *****
583 # *****
584 # *****
585 # *****
586 # *****
587 # *****
588 # *****
589 # *****
590 # *****
591 # *****
592 # *****
593 # *****
594 # *****
595 # *****
596 # *****
597 # *****
598 # *****
599 # *****
600 # *****
601 # *****
602 # *****
603 # *****
604 # *****
605 # *****
606 # *****
607 # *****
608 # *****
609 # *****
610 # *****
611 # *****
612 # *****
613 # *****
614 # *****
615 # *****
616 # *****
617 # *****
618 # *****
619 # *****
620 # *****
621 # *****
622 # *****
623 # *****
624 # *****
625 # *****
626 # *****
627 # *****
628 # *****
629 # *****
630 # *****
631 # *****
632 # *****
633 # *****
634 # *****
635 # *****
636 # *****
637 # *****
638 # *****
639 # *****
640 # *****
641 # *****
642 # *****
643 # *****
644 # *****
645 # *****
646 # *****
647 # *****
648 # *****
649 # *****
650 # *****
651 # *****
652 # *****
653 # *****
654 # *****
655 # *****
656 # *****
657 # *****
658 # *****
659 # *****
660 # *****
661 # *****
662 # *****
663 # *****
664 # *****
665 # *****
666 # *****
667 # *****
668 # *****
669 # *****
670 # *****
671 # *****
672 # *****
673 # *****
674 # *****
675 # *****
676 # *****
677 # *****
678 # *****
679 # *****
680 # *****
681 # *****
682 # *****
683 # *****
684 # *****
685 # *****
686 # *****
687 # *****
688 # *****
689 # *****
690 # *****
691 # *****
692 # *****
693 # *****
694 # *****
695 # *****
696 # *****
697 # *****
698 # *****
699 # *****
700 # *****
701 # *****
702 # *****
703 # *****
704 # *****
705 # *****
706 # *****
707 # *****
708 # *****
709 # *****
710 # *****
711 # *****
712 # *****
713 # *****
714 # *****
715 # *****
716 # *****
717 # *****
718 # *****
719 # *****
720 # *****
721 # *****
722 # *****
723 # *****
724 # *****
725 # *****
726 # *****
727 # *****
728 # *****
729 # *****
730 # *****
731 # *****
732 # *****
733 # *****
734 # *****
735 # *****
736 # *****
737 # *****
738 # *****
739 # *****
740 # *****
741 # *****
742 # *****
743 # *****
744 # *****
745 # *****
746 # *****
747 # *****
748 # *****
749 # *****
750 # *****
751 # *****
752 # *****
753 # *****
754 # *****
755 # *****
756 # *****
757 # *****
758 # *****
759 # *****
760 # *****
761 # *****
762 # *****
763 # *****
764 # *****
765 # *****
766 # *****
767 # *****
768 # *****
769 # *****
770 # *****
771 # *****
772 # *****
773 # *****
774 # *****
775 # *****
776 # *****
777 # *****
778 # *****
779 # *****
780 # *****
781 # *****
782 # *****
783 # *****
784 # *****
785 # *****
786 # *****
787 # *****
788 # *****
789 # *****
790 # *****
791 # *****
792 # *****
793 # *****
794 # *****
795 # *****
796 # *****
797 # *****
798 # *****
799 # *****
800 # *****
801 # *****
802 # *****
803 # *****
804 # *****
805 # *****
806 # *****
807 # *****
808 # *****
809 # *****
810 # *****
811 # *****
812 # *****
813 # *****
814 # *****
815 # *****
816 # *****
817 # *****
818 # *****
819 # *****
820 # *****
821 # *****
822 # *****
823 # *****
824 # *****
825 # *****
826 # *****
827 # *****
828 # *****
829 # *****
830 # *****
831 # *****
832 # *****
833 # *****
834 # *****
835 # *****
836 # *****
837 # *****
838 # *****
839 # *****
840 # *****
841 # *****
842 # *****
843 # *****
844 # *****
845 # *****
846 # *****
847 # *****
848 # *****
849 # *****
850 # *****
851 # *****
852 # *****
853 # *****
854 # *****
855 # *****
856 # *****
857 # *****
858 # *****
859 # *****
860 # *****
861 # *****
862 # *****
863 # *****
864 # *****
865 # *****
866 # *****
867 # *****
868 # *****
869 # *****
870 # *****
871 # *****
872 # *****
873 # *****
874 # *****
875 # *****
876 # *****
877 # *****
878 # *****
879 # *****
880 # *****
881 # *****
882 # *****
883 # *****
884 # *****
885 # *****
886 # *****
887 # *****
888 # *****
889 # *****
890 # *****
891 # *****
892 # *****
893 # *****
894 # *****
895 # *****
896 # *****
897 # *****
898 # *****
899 # *****
900 # *****
901 # *****
902 # *****
903 # *****
904 # *****
905 # *****
906 # *****
907 # *****
908 # *****
909 # *****
910 # *****
911 # *****
912 # *****
913 # *****
914 # *****
915 # *****
916 # *****
917 # *****
918 # *****
919 # *****
920 # *****
921 # *****
922 # *****
923 # *****
924 # *****
925 # *****
926 # *****
927 # *****
928 # *****
929 # *****
930 # *****
931 # *****
932 # *****
933 # *****
934 # *****
935 # *****
936 # *****
937 # *****
938 # *****
939 # *****
940 # *****
941 # *****
942 # *****
943 # *****
944 # *****
945 # *****
946 # *****
947 # *****
948 # *****
949 # *****
950 # *****
951 # *****
952 # *****
953 # *****
954 # *****
955 # *****
956 # *****
957 # *****
958 # *****
959 # *****
960 # *****
961 # *****
962 # *****
963 # *****
964 # *****
965 # *****
966 # *****
967 # *****
968 # *****
969 # *****
970 # *****
971 # *****
972 # *****
973 # *****
974 # *****
975 # *****
976 # *****
977 # *****
978 # *****
979 # *****
980 # *****
981 # *****
982 # *****
983 # *****
984 # *****
985 # *****
986 # *****
987 # *****
988 # *****
989 # *****
990 # *****
991 # *****
992 # *****
993 # *****
994 # *****
995 # *****
996 # *****
997 # *****
998 # *****
999 # *****
1000 # *****

```

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

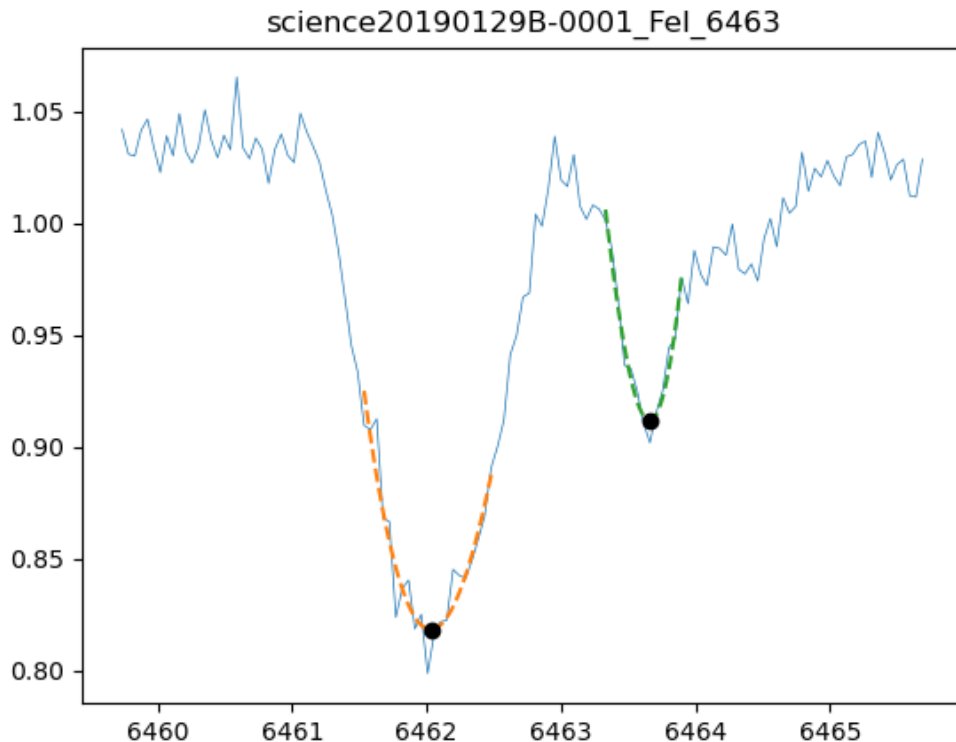


Figure 27.5: Grafische Darstellung der Minimumermittlung per Regressionskurve für die doppelte FeI 6463-Linie in einem Doppelsternspektrum (6 Tri A).

```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Liest einen Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im
5fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden)
6Koordinaten des Beobachters und Objekts die heliozentrische Korrektur,
7fitted die gewählte Linie in Minimumbereich per Regression
8und bestimmt aus dem heliozentrisch korrigierten Minimum die
9heliozentrisch korrigierte Radialgeschwindigkeit RV.
10Plottet alle fittings und gibt ermittelte Daten als ascii-Dateien
11(tab-separiert, als .dat) aus.
12
13Created on Mon Mar  8 13:43:44 2021
14
15@author: lothar
16"""
17
18import numpy as np
19from astropy.io import fits, ascii
20import glob
21import matplotlib.pyplot as plt
22from PyAstronomy import pyasl
23
24plt.style.use('seaborn-whitegrid')
25
26
27# ***** Koordinaten des Observatoriums in folgender Form: *****
28# longitude = 289.5967661 in Grad
29# latitude = -24.62586983 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 = 49.47527777777778
38# altitude = 280
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# # Koordinaten von del Cep
58# ra2000 = 22.48618473
59# dec2000 = +58.415198
60
61# Koordinaten von gam Cyg
62# ra2000 = 385.55798
63# dec2000 = +40.2566
64
65# Koordinaten von Betageuze
66# ra2000 = 88.7929583
67# dec2000 = 7.40785555
68
69# Koordinaten von thetai Ori C
70# ra2000 = 83.81858333
71# dec2000 = -5.389694444
72# *****
73
74# Fileliste erstellen. Spektren in einem (Unter)Ordner.
75files = input('Pfad und Name der Spektren (nutze wildcards): ')
76filelist = glob.glob(files)
77
78# Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
79# zeitliche Ordnung
80filelist.sort()
81
82# Ausdruck der Liste
83print('\nSpektralliste: \n')
84print('Anzahl der Spektren: ', len(filelist), '\n')
85
86
87# Liste der wählbaren Linien:
88Linien = {'CIV 7726': 7726.2,
89          'CIII 7837': 7837.25,
90          'HeI 6678': 6678.149,
91          'FeI 6678': 6677.9865,
92          'FeI 6634': 6633.7492,
93          'FeI 6609': 6609.1098,
94          'H_alpha': 6562.817,
95          'FeI 6546': 6546.24,
96          'FeII 6516': 6516.0783,
97          'FeI 6463': 6462.725,
98          'FeII 6456': 6456.3805,
99          'FeI 6417': 6416.9386,
100         'FeI 6412': 6411.6592,
101         'FeI 6408': 6408.0272,
102         'FeI 6400': 6400.0008,
103         'FeI 6394': 6393.6009,
104         'SiII 6371': 6371.36,
105         'SiII 6347': 6347.10,
106         'FeI 6265': 6265.1335,
107         'FeI 6256': 6256.3611,

```

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...
Öffnen  Speichern  Python 3  Tabulatorbreite: 4  Z. 8, Sp. 59  EINF

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'])

```

```

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.poly1d(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', 'BC', 'RV', 'RV_bc', 'miniflux'], format='tab')
324

```

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 eingengt, wodurch nur noch der untere Teil der Linie modelliert wird.


```

Öffnen  RV_MessungAnLinie_Zeitreihe_fits_perRBF.py  Speichern
268 suchintervall = int(1/step)
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 plt.plot(obs_time, RV, 'bo', markersize=1)
303
304 # Abspeichern als ascii-Datei
305
306 # Speichern von obs time und corr in ascii-file
307
308 # Speichern von obs time und hjd in ascii-file
309
310 # Speichern von obs time und corr in ascii-file
311
312
313
314 # Speichern von obs time und corr in ascii-file
315
316 # Speichern von obs time und hjd in ascii-file
317
318 # Speichern von obs time und hjd in ascii-file
319
320 # Speichern von obs time und hjd in ascii-file
321
322

```

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

```

Öffnen  RV_MessungAnLinie_Zeitreihe_fits_perSpline.py  Speichern
261 # Breite des Suchintervalls, Breite in Angström, anpassen.
262 suchintervall = int(1/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('baryzentrisch 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 plt.plot(obs_time, RV, 'bo', markersize=1)
295
296 # Abspeichern als ascii-Datei
297
298 # Speichern von obs time und corr in ascii-file
299
300 # Speichern von obs time und hjd in ascii-file
301
302 # Speichern von obs time und corr in ascii-file
303
304 # Speichern von obs time und hjd in ascii-file
305
306 # Speichern von obs time und corr in ascii-file
307
308 # Speichern von obs time und hjd in ascii-file
309
310 # Speichern von obs time und hjd in ascii-file
311
312 # Speichern von obs time und hjd in ascii-file
313
314

```

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 *Zeitserie_EW.py* erfolgen (Code in Abb. 29.1). Der Integrationsbereich wird interaktiv per Mausklick 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 + refix - 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]})
319
320 gf2 = fuf.GaussFit1d()
321 gf2.setRestriction({'A': [None, 0]})
322 gf2['A'] = gf1['A']
323 gf2['sig'] = gf1['sig']
324 gf2['off'] = gf1['off']
325 gf2['mu'] = gf1['mu']
326 gf2.thaw(['A', 'sig', 'off', 'mu'])
327 gf2.fit(intervallwave2, intervallflux2)
328 # gf2.parameterSummary()
329 linienminimum2[i] = gf2['mu']
330 if linienminimum2[i] <= wellenlaenge - 4:
331     break
332 if linienminimum2[i] >= wellenlaenge + 4:
333     break
334 linienminimum2_bc[i] = linienminimum2[i] * (1 + corr[i]/299792)
335 RV2[i] = (linienminimum2_bc[i] - wellenlaenge)/linienminimum2_bc[i]*299792
336
337 plt.plot(intervallwave2, gf2.model, 'b--')
338 RV_diff1[i] = RV2[i] - RV1[i]
339
340 # dritter Durchlauf, eingegrenzt auf Intervall 2*sigma
341 mu_index = int((gf2['mu']-lambda0)/step + refix - 1)
342 suchintervall3 = abs(int(1.2*gf2['sig']/step))
343 intervallflux3 = np.zeros(suchintervall3)
344 intervallwave3 = np.zeros(suchintervall3)
345 for k in range(len(intervallflux3)):
346     intervallflux3[k] = flux[k + mu_index - suchintervall3//2]
347     intervallwave3[k] = lambda0 + (k + mu_index - suchintervall3//2)*step
348 gf3 = fuf.GaussFit1d()
349 gf3.setRestriction({'A': [None, 0]})
350 gf3['A'] = gf2['A']
351 gf3['sig'] = gf2['sig']
352 gf3['off'] = gf2['off']
353 gf3['mu'] = gf2['mu']
354 gf3.thaw(['A', 'sig', 'off', 'mu'])
355 gf3.fit(intervallwave3, intervallflux3)
356 # gf3.parameterSummary()
357 linienminimum3[i] = gf3['mu']
358 if linienminimum3[i] <= wellenlaenge - 4:
359     break
360 if linienminimum3[i] >= wellenlaenge + 4:
361     break
362 linienminimum3_bc[i] = linienminimum3[i] * (1 + corr[i]/299792)
363 RV3[i] = (linienminimum3_bc[i] - wellenlaenge)/linienminimum3_bc[i]*299792
364
365 plt.plot(intervallwave3, gf3.model, 'g--')
366 plt.title(filelist[i] + ' Beobachtungszeitpunkt ' + str(obs_time[i]))
367 plt.plot(linienminimum3[i], apex3[i], 'o', color='black')
368
369 RV_diff2[i] = RV3[i] - RV2[i]
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 hjd in ascii-file
399 # ascii.write([obs_time, hjd], linie='Tabelle_obs_time_hjd.dat', overwrite=True,
400 #             names=['JD', 'HJD'], format='tab')

```

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"""
4Berechnet für eine Zeitserie im fits-Format die Äquivalentweite einer Serie.
5Eingabe der Integrationsgrenzen grafisch-interaktiv oder manuell.
6Die EW-Berechnung setzt voraus, dass für alle Spektren der Serie das gleiche
7Wellenlängenintervall für die Berechnung des Integrals verwendet werden kann,
8also keine wesentlichen RV-Änderungen stattfinden, oder wenn doch, dass die
9Linie isoliert ist (also der Flux = 1 im Umfeld ist).
10
11Created on Fri Oct 16 17:47:58 2020
12
13@author: lothar
14"""
15
16import numpy as np
17from astropy.io import fits, ascii
18import matplotlib.pyplot as plt
19import glob
20
21plt.ion()
22
23# Fileliste erstellen. Spektren in einem (Unter)Ordner.
24files = input("Pfad und Name der Spektren (nutze wildcards): ")
25filelist = glob.glob(files)
26
27# Alphabetisches Sortieren. Bei richtiger Namensgebung ergibt das eine
28# zeitliche Ordnung
29filelist.sort()
30
31# Ausdruck der Liste
32print("\nSpektralliste: \n")
33print(filelist)
34print("Anzahl der Spektren: ", len(filelist), "\n")
35
36EW_dict = {}
37
38k = 0
39sp = fits.open(filelist[k], ignore_missing_end=True)
40print("\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
43flux = np.array(sp[0].data)
44wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
45
46for 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["CDELT1"]
50    )
51    # The list wave contains the wavelengths of the pixels.
52    # Close the fits-file
53sp.close()
54
55# Plot the spectrum

```

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

```

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
14author: 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
34thead_flag = input('Möchten Sie den header des template komplett sehen? \
35                    /n:')
36if theader_flag == '/n':
37    print('Headerdaten des template:')
38    print(theader)
39# Prüfung auf die nötigen header-Einträge
40print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
41if 'NAXIS1' in theader:
42    print('Dimension, NAXIS1: ', theader['NAXIS1'])
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 'CDELT1' in theader:
56    tcdel = theader['CDELT1']
57    print('Schrittweite der Wellenlänge, CDELT1: ', tcdel)
58else:
59    print('CDELT1 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*tcdel
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: Kreuzkorrelationsskript 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.

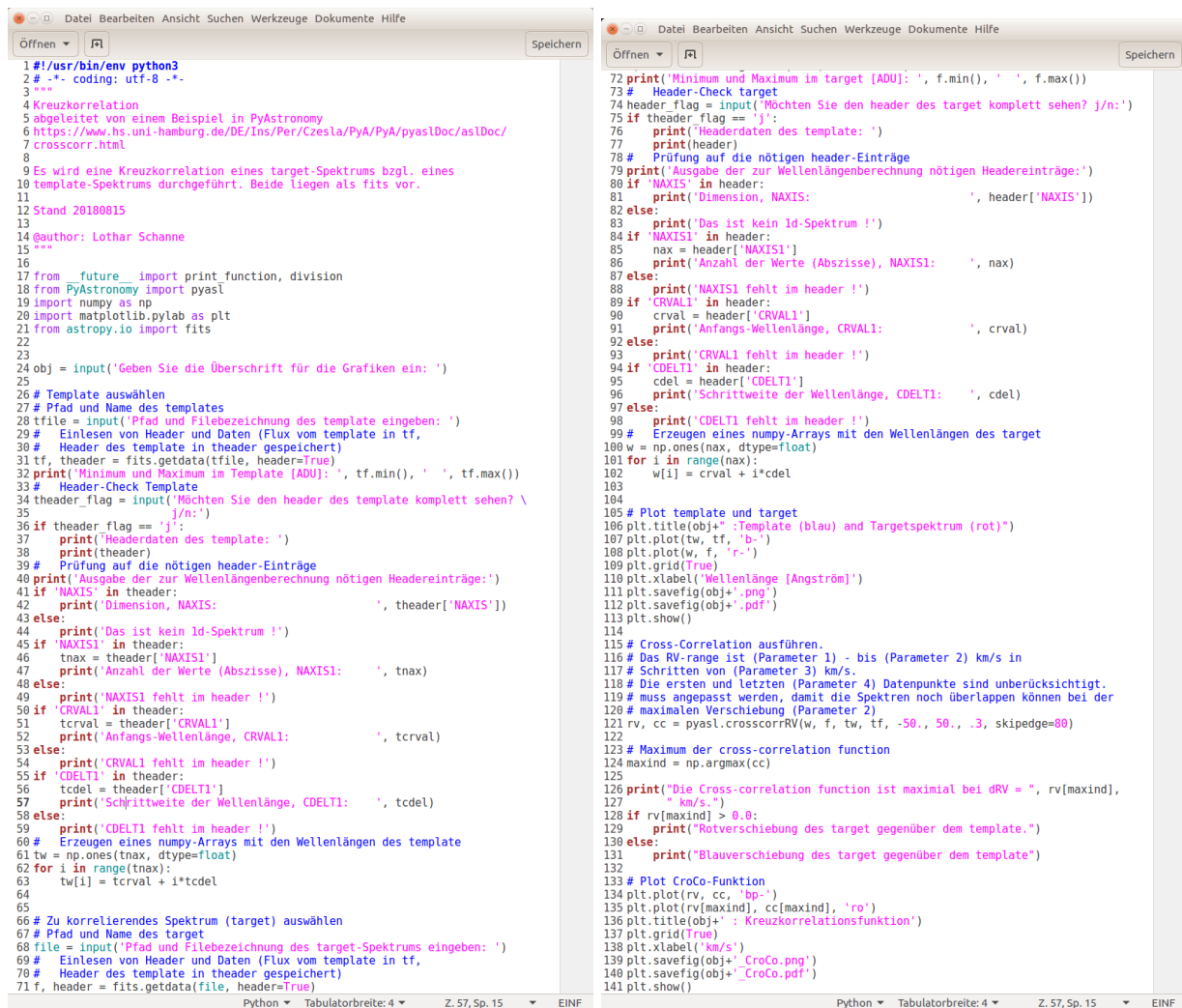


Figure 31.1: Kreuzkorrelationsskript für 1d-ascii-Spektren.

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 kreuzkorreliert.

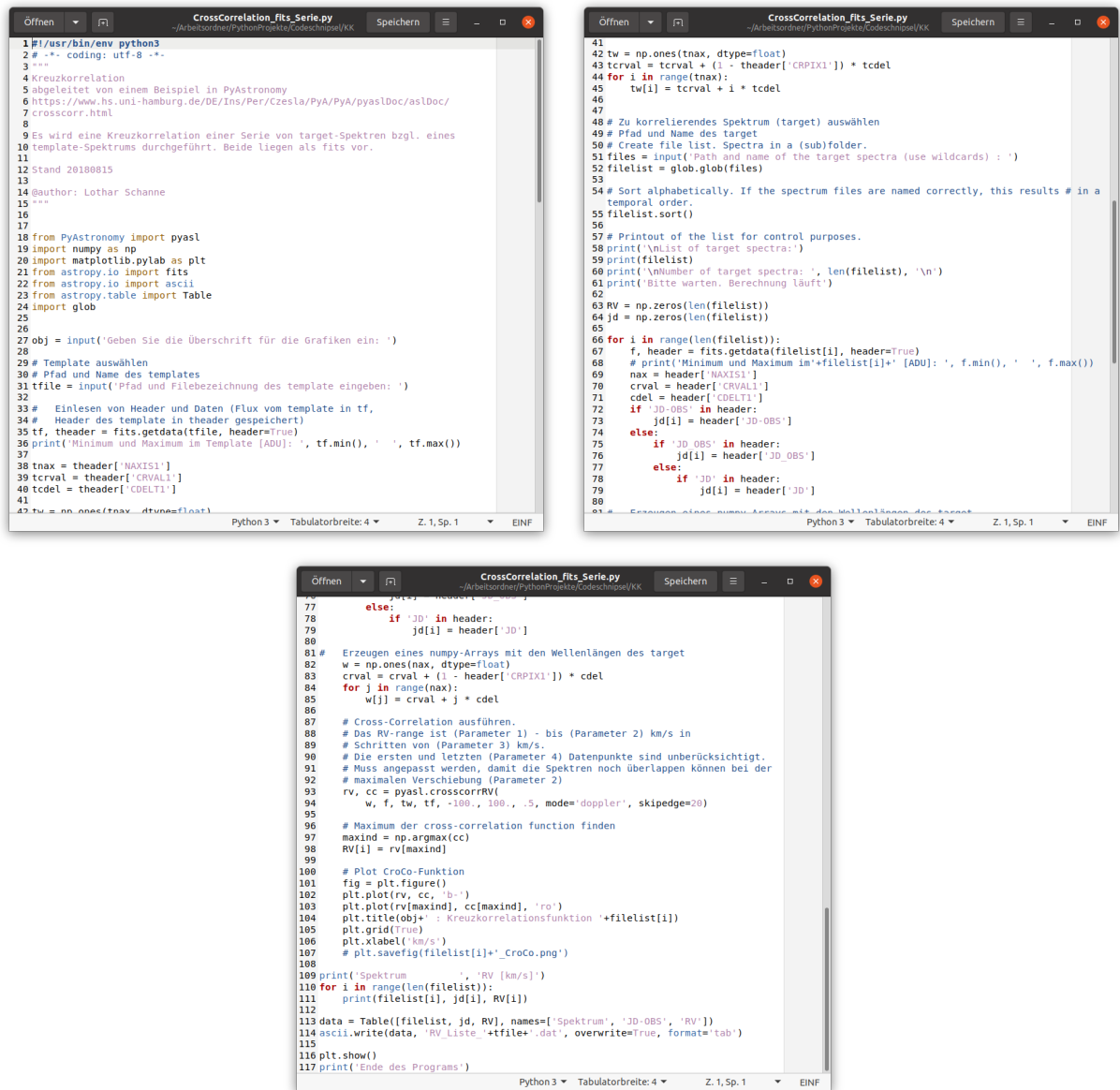


Figure 32.1: Kreuzkorrelationsskript für eine Spektrenserie im fits-Format.

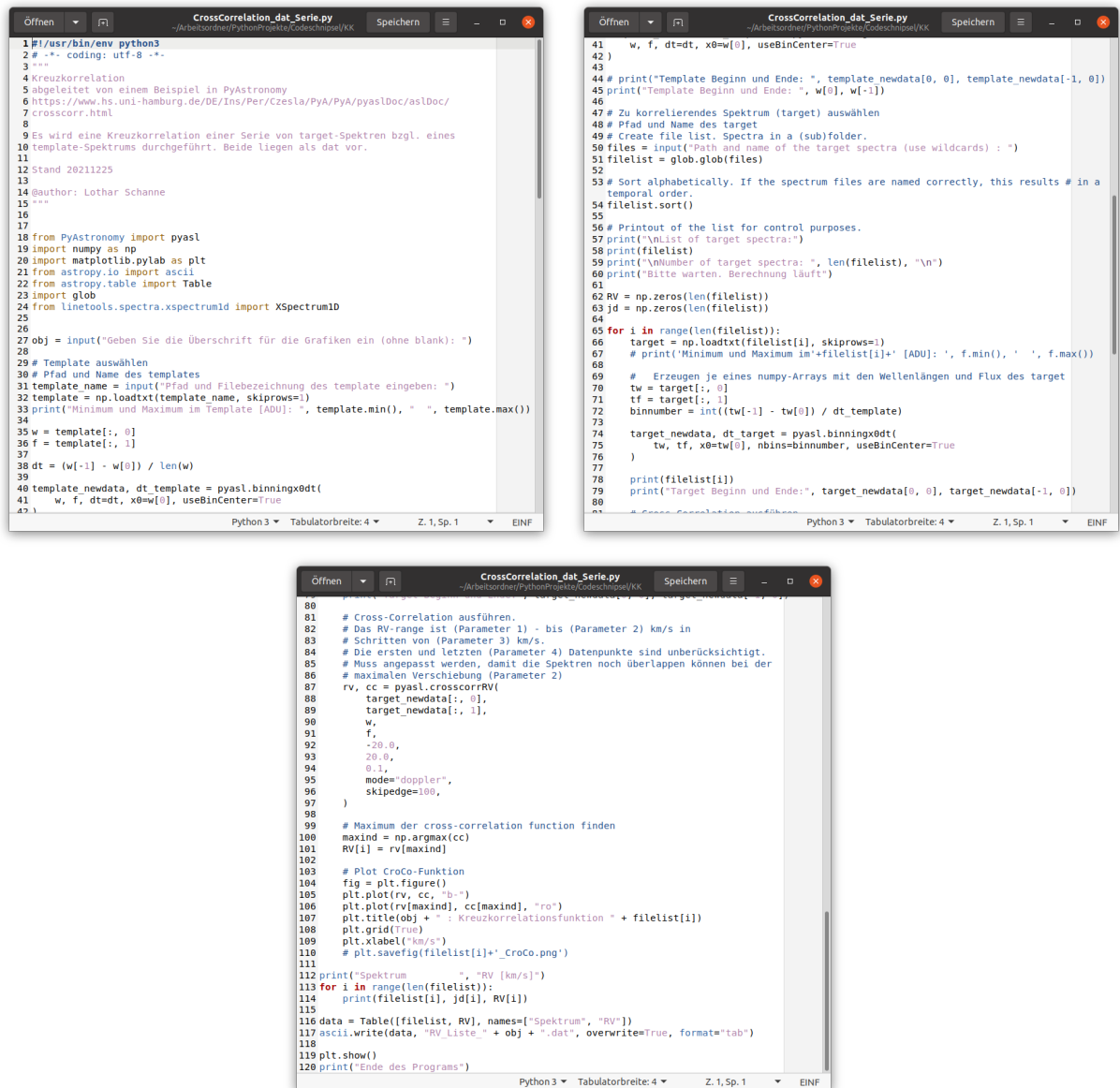


Figure 33.1: Kreuzkorrelationsskript für eine Spektrenserie im ascii-Format.