Spectro-Python

Auswertung optischer astronomischer Spektren mit Hilfe von Python 3

April 1, 2024

Contents

I. Motivation für die Verwendung der Interpretersprache Python 3

II. Installation einer persönlichen Python-Version 3

1. Allgemeines 3
1.1. Anmerkung: 4

2. Installation einer Python-Version im eigenen User-Verzeichnis 4
2.1. Eine völlig unabhängige Pythonversion im eigenen User-Verzeichnis erstellen 4
2.2. Ein Environment des betriebssystemeigenen Pythons für Astroanwendungen in Python erstellen 5

3. Installation einer Entwicklungsumgebung (Spyder) und SciPy (wissenschaftliches Rechnen) 5

4. Verwendung von Spyder 6

5. Zur Einführung ein erstes spektroskopisches Programm 6

III. Manipulation von 1d-fits-Dateien (1d-Spektren) 11

6. Kontrolle und Korrektur des Headers einer Serie von 1d-Spektren im fits-Format 11

7. Erzeugen eines definierten Wellenlängenausschnitts aus einem fits-1d-Spektrum 12

8. Übersetzen eines 1d-Spektrums im fits-Format in ascii-Dateien 12

IV. Zeitreihen von 1d-Spektren 12

9. Zeitreihen von 1d-Spektren im fits-Format grafisch darstellen 12

V. Umgang mit 1d-Spektren im ASCII-Format 18

10. Plotten eines ascii-1d-Spektrums 18

11. Plotten und erzeugen eines Ausschnitts aus einem ascii-1d-Spektrums 18

VI. Konvolution (Faltung) von Modell-1d-Spektren mit Gaussfunktionen 18

12. *Pollux_Ausschnitt_convol.py* 21

13. *convol_dat.py* 23

14. Vergleich von theoretischem und gemessenen Spektrum (overplot) 23

VII. Normierung von 1d-Spektren 23

VIII. Baryzentrische Korrektur (barycentric correction BC) 28

15. Berechnung der BC 29

16. BC-Korrektur einer Serie von 1d-Spektren durch Dopplerverschiebung 29

IX. Kreuzkorrelationen (KK) 30

17. KK für 1d-Spektren des Dateityps fit 30

18. KK für 1d-Spektren im ASCII-Format 32

Part I.
Motivation für die Verwendung der Interpretersprache Python

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-Midas[1] und IRAF[2] 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 IRIS[3] und VSpec[4], RSpec[5] und BASS[6].

Die Autor verwendegt auf einer Linux-Plattform (Debian 12) 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 wird die Interpretersprache *Python* empfohlen.

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 Modulen im Internet, die 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. Ich selbst verwendete für den Einstieg in Python ein Buch von Bernd Klein, *Einführung in Python* in der 3. Auflage[7], als Nachschlagewerk den Klassiker von Mark Lutz[8] und für Spezialitäten (numpy, pandas, matplotlib) das Buch *Data Science* mit Python[9]

Part II.
Installation einer persönlichen Python-Version

1. Allgemeines

Python gibt es in unterschiedlichen Distributionen und Versionen[10]. Beispielsweise ist die veraltete, aber immer noch funktionierende Version Python 2 derzeit im letzten Release 2.7 vorhanden, wird

[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/
[6] http://aesesas.com/mediapool/142/1423849/data/DOCUMENTOS/BASS_Project_1_.pdf
[7] Hanser Verlag, 2018.
[8] Learning Python, 5th Edition Powerful Object-Oriented Programming
By Mark Lutz
Publisher: O'Reilly Media
Release Date: June 2013
Pages: 1648
[9] Autor: VanderPlas, Jake von mitp Verlag 549 Seiten, Softcover ersch. 01/2018 ISBN: 978-3-95845-695-2
[10] https://www.python.org/

aber nicht mehr weiter entwickelt. Die neueren Python-Distributionen sind in Python 3 geschrieben, die aktuelle stabile Version ist derzeit Python 3.13.

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.xy'*“ ein neuer Modul eingespielt wird, kann es passieren, dass vom Betriebssystem noch 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 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 im eigenen Userverzeichnis zu installieren: Anaconda[11]. Die kostenlose Version enthält über 1400 data science packages und verwaltet alle Hintergrundbibliotheken und -skripte automatisch mit dem eigenen Verwaltungsprogramm *conda*. Außerdem sind die wichtigen Bibliotheken und Programme wie Spyder, SciPy, NumPy, Pandas, Matplotlib bereits mitinstalliert. Diese recht vollständige Python-Distribution sei Python-Einsteigern dringend empfohlen. Sie hat aber den Nachteil, dass nicht alle benötigten Pythonpakete mit *conda* zugänglich sind und dann diese Pakete mit *pip* installiert werden müssen, wodurch es Probleme mit der Paketverwaltung geben kann.

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

Möchte man unabhängig von einer Firma wie anaconda sein eigenes Python installieren, sei eine der folgenden Vorgehensweisen empfohlen.

2.1. Eine völlig unabhängige Pythonversion im eigenen User-Verzeichnis erstellen

Von der Internetseite der python.org [12] lädt man sich 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 Usage[13]. 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 ist der ganze Installationsvorgang vollständig beschrieben auf einer Webpage[14]. 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

[11] https://www.anaconda.com/
[12] https://www.python.org/
[13] https://docs.python.org/3/using/index.html
[14] https://wiki.ubuntuusers.de/Python/manuelle_Installation/

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* (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 werden[15].

Wenn alle diese Tipps berücksichtigt wurden hat man innerhalb einiger Minuten bis einiger Stunden[16] 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, diee hier weiter unten aufgeführt werden.

2.2. Ein Environment des betriebssystemeigenen Pythons für Astroanwendungen in Python erstellen

In Linux-Systemen ist immer eine eine systemeigene Pythonversion installiert, die vom Betriebssystem (z.B. Debian12 oder Ubuntu) verwendet wird. Es ist nun möglich, im eigenen User-Verzeichnis (/home/user) eine Umgebung (environment) dieser Pythonversion zu installieren, die dann nur für die eigenen Astroanwendungen von Python benutzt wird, wodurch die betriebssystem-eigene Version immer unverändert dem Betriebssystem vorbehalten bleibt. Also Änderungen im Environment beeinflussen (und stören) nicht die Betriebsystemversion.

Dazu muss man mit sudo (also mit Administratorrechten) das Paket venv installieren (wenn es nicht schon standardmäßig installiert ist) und anschließend das eigene Verzeichnis erstellen, in meinem Falle mit dem Befehl

python3 -m venv ~/astro-env

wobei die Tilde ~ das eigene home-Verzeichnis des jeweiligen Users bedeutet. Dadurch wird im eigenen Home-Verzeichnis ein Verzeichnis astro-env erzeugt, das das environment enthält.

Um nun astrospezifische Pythonpakete in diesem environment zu installieren oder mit diesen zu arbeiten muss man zuvor in einer Konsole den Befehl

source astro-env/bin/activate

eingeben und dann mit

pip install paketname

das Pythonpaket 'paketname' installieren oder durch Eingabe des Programmnamens (z.B. *spyder*) das Python-Programm starten.

Mit dieser Variante des Umgangs mit Python hat der Verfasser die besten Erfahrungen gemacht.

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

Die Entwicklung von Programmen (Skripten) geht am bequemsten mit einer Entwicklungsumgebung wie IDLE (in Python 3 integriert) oder *Spyder*. Letzteres bevorzugt der Autor. Um es zu Verfügung zu haben sollte ein Zusatzmodul installiert werden namens *jupyter*. Es ist Bestandteil eines großen Pakets

[15] Man verlässt Python in dem man quit() eingibt.
[16] Die Quellcodes werden alle kompiliert, was je nach Schnelligkeit des eigenen PC/Laptops sehr unterschiedlich lange dauert.

von wissenschaftlichen Zusatzbibliotheken, die von der community der SciPy.org entwickelt werden[17]. 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* (Matrizenrechnung 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 *spyder* die Entwicklungsumgebung aufrufen und damit arbeiten. Alternativ auch mit den für das Betriebssystem üblichen Programmstartroutinen.

Das Paketverwaltungsprogramm *pip* muss konsequent angewendet werden. Entweder alles mit pip installieren oder nichts, sonst kann es Durcheinander mit den Hintergrundskripten geben. Hat man die Anaconda-Distribution installiert, darf 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 bleibt.

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

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 ausgeführt werden. Die Ergebnisse werden unmittelbar in der IPython-Konsole angezeigt. Bzgl. der detaillierten Beschreibung der Arbeitsweise mit Spyder sei auf die Hilfe (Help) im Menü oben oder die Spyder-Dokumentation verwiesen.

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

5. Zur Einführung ein erstes spektroskopisches Programm

In den folgenden Abschnitten werde einige Pythonskripte vorgestellt, welche die Verwendungsmöglichkeiten selbst erstellter Programme im Rahmen der spektroskopischen Arbeit demonstrieren sollen.

Als erste Aufgabe wollen wir uns ein bereits reduziertes 1d-Spektrum im üblichen .fit-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 im verwendeten Python installiert sind (import-Anweisungen am Anfang des Python-Skripts).

[17] https://scipy.org/

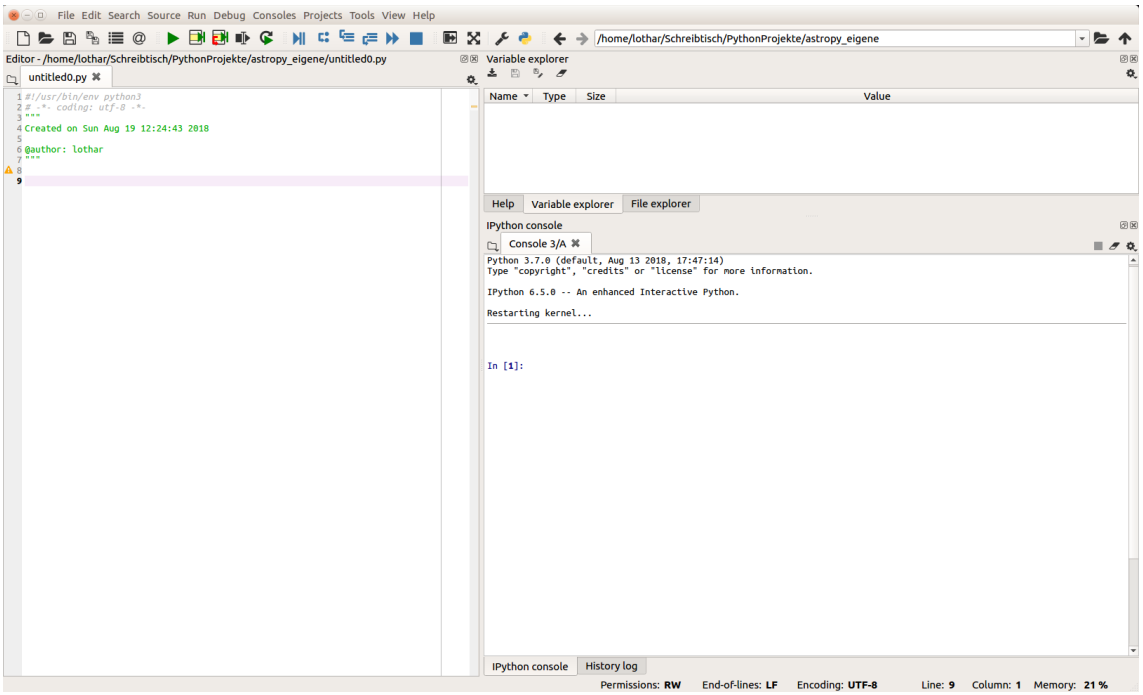


Figure 4.1: Spyder-Fenster nach dem Start von Spyder.

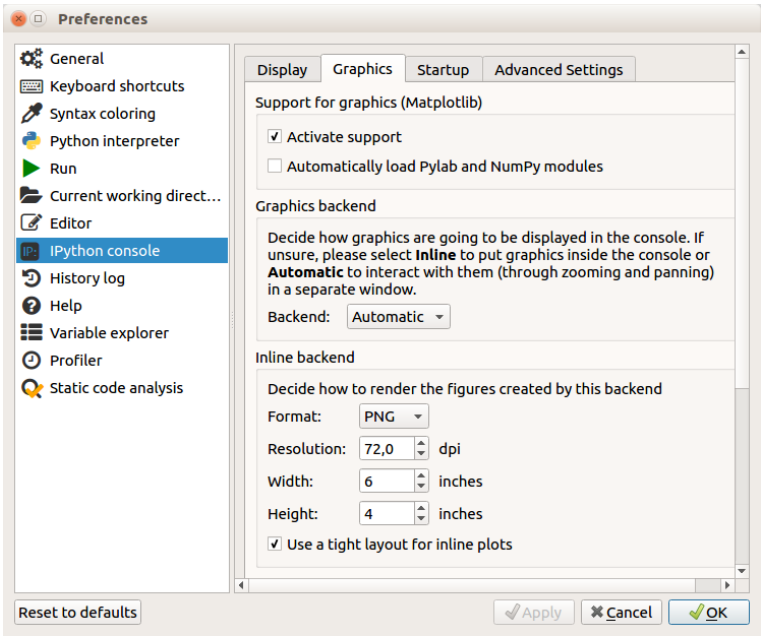


Figure 4.2: Empfohlene Einstellungen für die IPython-Konsole in Spyder.

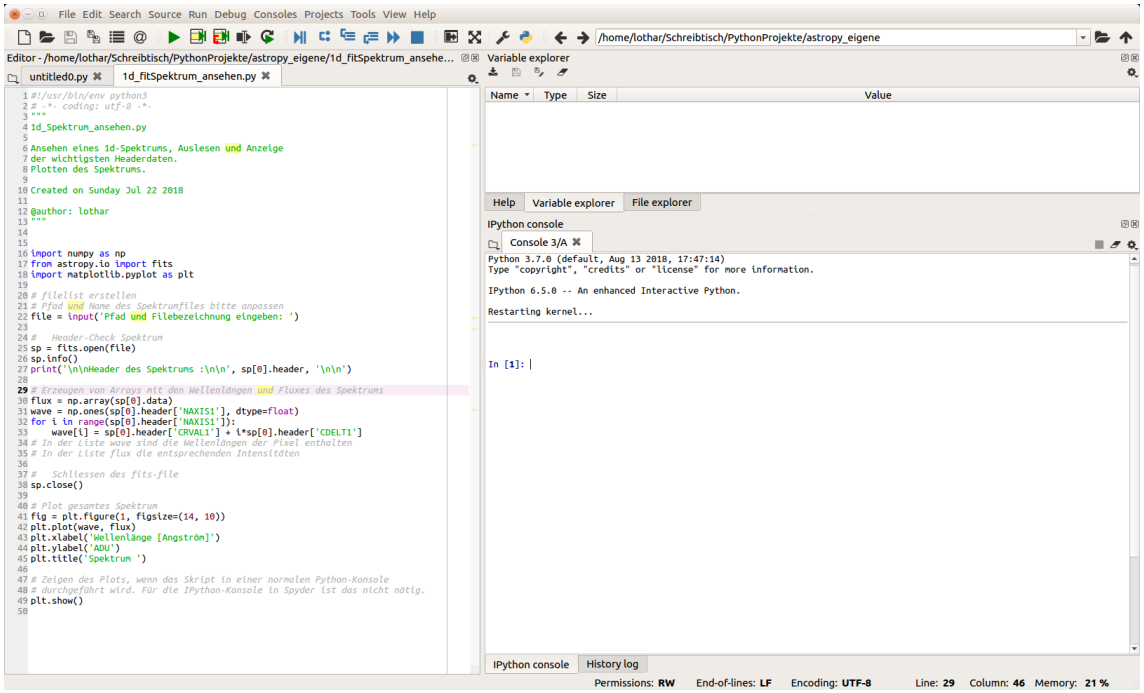


Figure 5.1: Programm zur Ansicht eines 1d-Spektrums im fits-Format und der Header-Angaben, dargestellt in der Entwicklungsumgebung Spyder, linkes Fenster.

In Abb. 5.1 sehen wir im linken Fenster das Skript *1d_fitSpekrum_ansehen.py*. Dies ist ein einfacher Textfile, der in jedem Editor geschrieben, angezeigt und manipuliert werden kann. Dass es ein Pythonprogramm ist drücken wir mit der Extension *.py* aus. Sehen wir uns diesen Text an. In grün 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. Hier werden die Module (= Programmpakete) aufgerufen, die folgendes bewirken:

- *numpy* ist ein Modul zur schnelleren Verarbeitung numerischer Daten[18]. Es stellt insbesondere Strukturen zu Verfügung, in denen Daten gespeichert und anschließend schnell umgruppiert, geändert und verarbeitet werden können. numpy erhält die Abkürzung *np* und wird unter diesem *Alias (Spitznamen)* verwendet.

- astropy.io ist ein Bestandteil des astronomischen Pakets Astropy[19]. Es dient zum Ein- und Auslesen von Daten, z.B. fits-Files. Dieses Paket muss natürlich zuerst in unserem Python3 installiert werden, bevor wir es importieren und verwenden können. Siehe Angaben unter der Fußnote (*pip install astropy*). Hier wird nur das Paket *fits* aus *astropy.io* importiert, weil wir nicht mehr brauchen.

- Das Modul *matplotlib* enthält eine wahre Fundgrube von grafischen Möglichkeiten[20]. Auch 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 String-Variable namens *file*, der wir den String (=Zeichenfolge) zuordnen, den wir in der Konsole nach der Aufforderung 'Pfad und Filebezeichnung eingeben:' manuell eingeben.

[18] http://www.numpy.org/
[19] http://www.astropy.org/
[20] https://matplotlib.org/

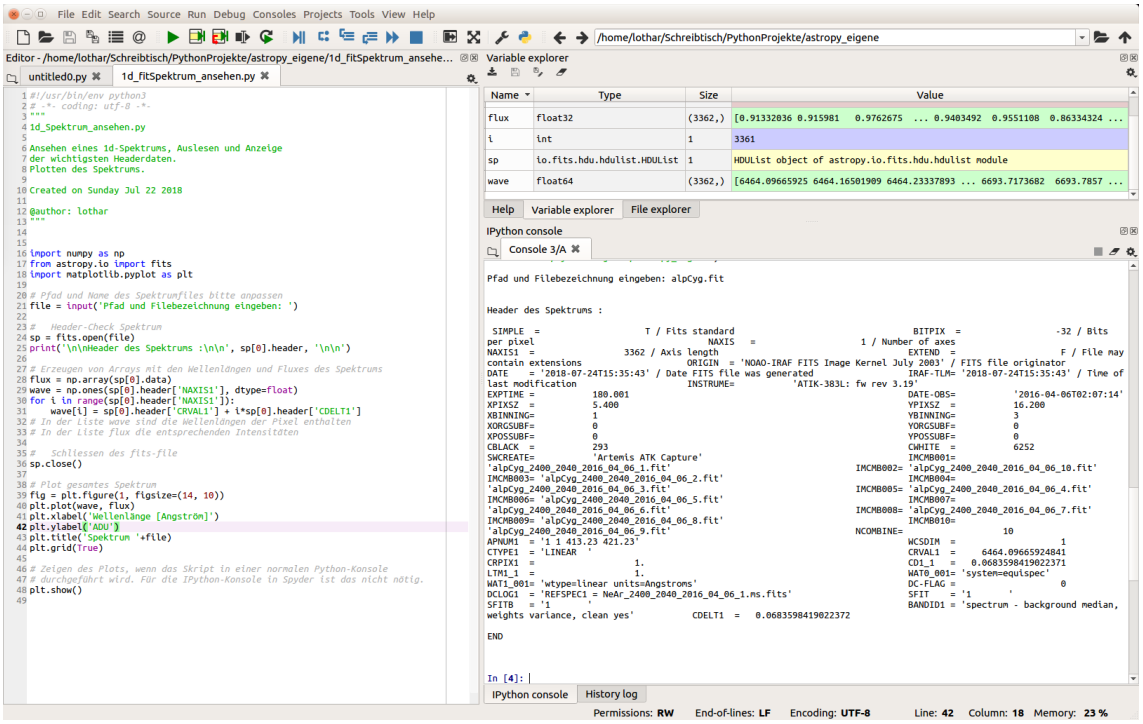


Figure 5.2: Ausdruck des Headers in der Konsole nach Start des Programms aus dem Editor heraus.

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 etc. Dazu müssen wir zuerst das File öffnen (Zeile 24). Hier sehen wir die typische Formulierung einer objektorientierten Sprache. Wir rufen aus der Instanz fits (die wir ja in Zeile 17 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 (Namen), hier *sp*.

In der nächsten Zeile 25 veranlassen wir das Programm, den Header von sp in der Konsole auszudrucken (mit dem Befehl *print*()). Das 1d-Spektrum enthält nur eine Dimension (entlang der Dispersion), diese wird durch sp[0] zugänglich. Und da wir den header sehen wollen ergibt die Methode sp[0].header genau den kompletten Inhalt des Headers, der nach der manuell programmierten Erläuterung 'Header des Spektrums :' in der Konsole ausgedruckt wird. Lassen wir das Programm aus dem Editor heraus in der Konsole laufen (Taste F5 oder grüner Pfeil), ergibt sich der Headerausdruck wie in Abb. 5.2. Alle Keywords des Headers sind in der Konsole rechts unten ausgegeben. Und auch alle Variablen werden im Variablenexplorer aufgeführt, inkl. Variablentyp, -größe und die ersten Werte dazu. Wichtig für die weiteren Berechnungen sind die Headerwerte:

1. NAXIS1 = 3362 / Axis length = Anzahl der Pixel/Wellenlängenpunkte
2. CRVAL1 = 6464.09665924841 = Start der Wellenlängenskala in Angström
3. CDELT1 = 0.0683598419022372 = Schrittweite, also das Wellenlängenintervall der einzelnen Pixel.

Mit den drei Größen lässt sich jedem Pixel die Wellenlänge zuordnen.[21].

[21] Die fits-Dateien enthalten neben dem Header die eigentlichen Daten, hier in in sp[0].data. Das ist einfach eine Zeile mit NAXIS1 Werten, nämlich den Spektrumintensitäten (Flux). Mit den obigen 3 Headerwerten lassen sich dann die zugehörigen Wellenlängen berechnen. Beispielsweise für das 100te Pixel durch
Wellenlänge[99] = CRVAL1 + 99 * CDELT1
Die Indexzählung beginnt in Python mit 0, also ist Wellenlänge[0] = CRVAL1 + 0 * CDELT1 die Startwellenlänge des Spektrums auf der kurzwelligen Seite.

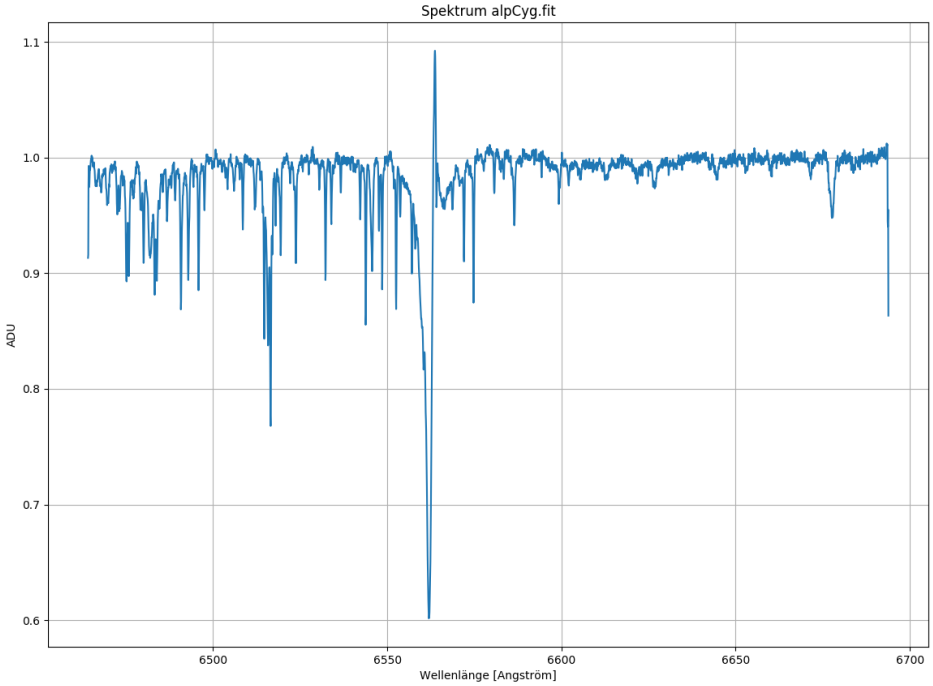


Figure 5.3: Plot des Spektrums.

Rechts oben in Abb. 5.2 sind im *Variable explorer* nach diesem Programm-Durchlauf die bereits berechneten Variablen aufgeführt. Hierauf sollte man immer einen Blick werfen und die Variablen auf Plausibilität, Vollständigkeit und den richtigen Typ prüfen.

Im weiteren Verlauf des Programmtextes beginnen wir in Zeile 28 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 3362 Fluxwerte. Diese übergeben wir in ein numpy-Objekt, nämlich ein numpyarray (eine spezielle Liste) und nennen diese Variable *flux*. In Zeile 29 erzeugen wir ein numpyarray namens *wave* mit lauter Einsen (1.), und zwar mit genau NAXIS1 (hier 3362) Stück. Dieses füllen wir im weiteren Verlauf mit den Wellenlängen der Pixel auf. Dazu bilden wir in der Schleife in Zeilen 30 und 31 Pixel für Pixel die Wellenlänge und speichern sie im wave-Element i ab. Nachdem das 3362 mal gemacht wurde, ist unser Array *wave* mit den Wellenlängen gefüllt (mit Zahlen im float-Format, also Gleitkommazahlen).

In Zeile 36 schließen wir das fits-File, wir benötigen es nicht mehr.

Dann beginnen wir in Zeile 39 den Spektrumplot zu programmieren. Dies geschieht nach den Regeln, wie sie in der Dokumentation von matplotlib beschrieben sind[22].

Lassen wir das Programm nochmal durchlaufen und betrachten den Plot des Spektrums, das unser Programm nun erzeugt hat. Beachten Sie, dass wir die Einstellungen für die IPython-Konsole 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 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.

[22] https://matplotlib.org/api/pyplot_summary.html, https://matplotlib.org/resources/index.html

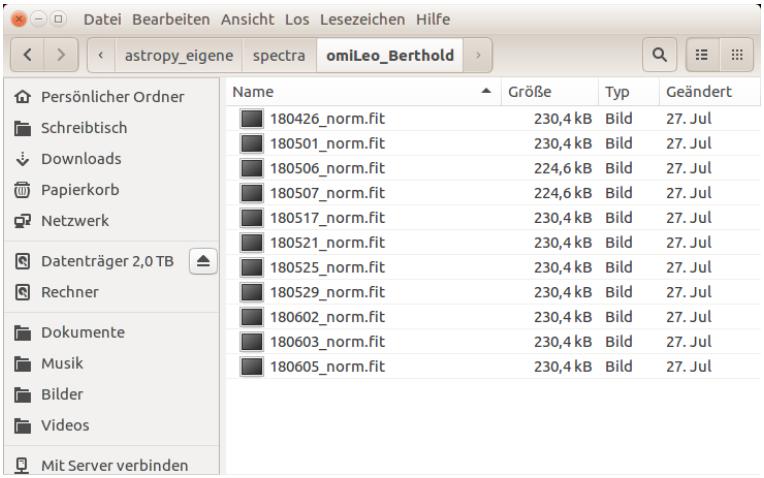


Figure 6.1: Zeitserie von normierten 1d-Echellespektren von omi Leo.

Part III.
Manipulation von 1d-fits-Dateien (1d-Spektren)

6. 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 Datenaquisitionssoftware, 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, inklusive des Beobachtunsgdatums aufgebaut sind. Ein Beispiel ist in Abb. 6.1 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. 6.2 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*[23] und wieder *fits* aus *astropy.io*. Grafische Hilfsmittel (matplotlib) sind diesmal nicht erforderlich.

Aus dem Pfad zu den Dateien (durch wildcards ausgewählt) wird eine Liste der Dateien erstellt, die in Zeile 22 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 weitere Auswertungen der Zeitserie vollständig und richtig sind.

Ab Zeile 39 bis 58 wird eine Headerlesefunktion definiert, welche eine Auswahl der Headerdaten

[23] *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, in https://pypi.org/ zu finden.*

zurückgibt. Mit den Erkenntnissen aus dem Header der ersten Datei lässt sich die Funktion vor einem zweiten Programm-Durchlauf bzgl. der rückzugebenden Headerdaten anpassen.

Nachdem die Funktion angepasst ist, wird die ausgelesene Headerdatenauswahl ab Zeile 62 tabellarisch in der Konsole ausgegeben. Falls Einträge nicht den Wünschen entsprechen, können sie geändert werden. Das geschieht ab Zeile 69. Der User wird gefragt, ob er Änderungen durchführen möchte. Wenn ja werden die Headerdaten geändert und anschließend die fits-Dateien entsprechend geändert (Zeile 76). Die Headerobjekte in den Zeilen 74 und 75 sind entsprechend anzupassen, zu löschen (beispielsweise durch auskommentieren) oder um weitere zu erweitern.

Die Headerdaten werden nach der Änderung zur Kontrolle nochmal in der Konsole tabellarisch ausgegeben. Der Endzustand nach dem Durchlaufen des Programms in Spyder ist in Abb. 6.3 gezeigt.

7. 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 kann.

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 neuen 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).

Das Programm ist in 7.1 vorgestellt.

8. 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).

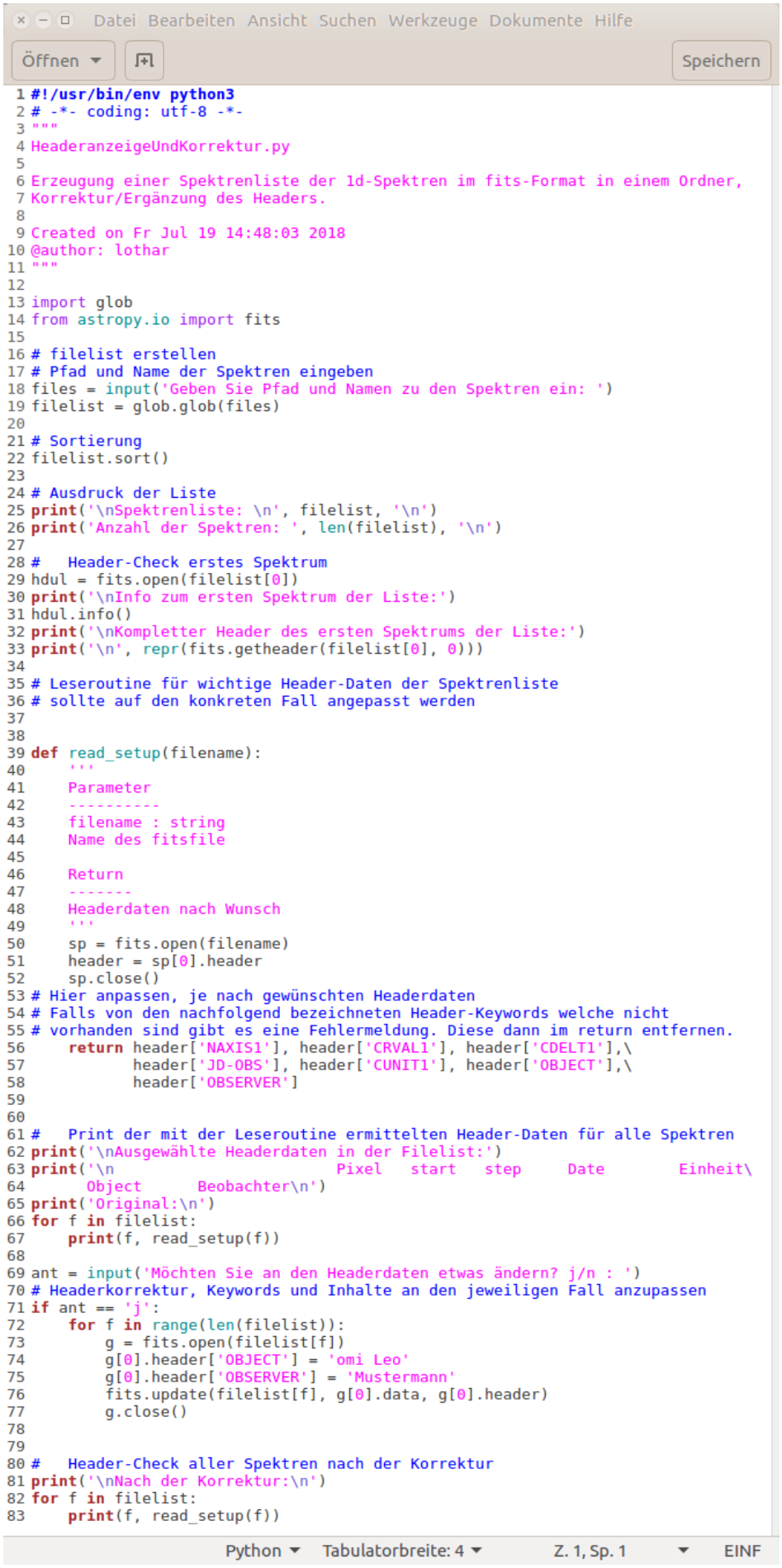
Das leistet das Python3-Skript *„fit_ in_ csv_ und_ dat.py“*. In Abb. 8.1 ist das Programmlisting gezeigt. In der erzeugten dat- und csv-Datei gibt es zwei Spalten von Fließkommazahlen (type float mit Punkt als Dezimalzeichen), überschrieben mit 'WAVE' und 'FLUX'.

Part IV.
Zeitreihen von 1d-Spektren

9. Zeitreihen von 1d-Spektren im fits-Format grafisch darstellen

Nach längerer Beobachtung von Objekten verspürt 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. 9.1 zu sehen. Die fits-Dateien werden aus einem Ordner eingelesen, ein Wellenlängenbereich gewählt und dann werden die beschnittenen 1d-Dateien als neue fits abgespeichert. In einem Plot werden sie übereinander und in in einem zweiten übereinander, aber senkrecht um einen wählbaren Betrag versetzt geplottet. Die Grafik wird auch als PDF und PNG abgespeichert.

In Abb. 9.2 ist ein Plot aus 11 Spektren gezeigt.




```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
HeaderanzeigeUndKorrektur.py

Erzeugung einer Spektrenliste der 1d-Spektren im fits-Format in einem Ordner,
Korrektur/Ergänzung des Headers.

Created on Fr Jul 19 14:48:03 2018
@author: lothar
"""

import glob
from astropy.io import fits

# filelist erstellen
# Pfad und Name der Spektren eingeben
files = input('Geben Sie Pfad und Namen zu den Spektren ein: ')
filelist = glob.glob(files)

# Sortierung
filelist.sort()

# Ausdruck der Liste
print('\nSpektrenliste: \n', filelist, '\n')
print('Anzahl der Spektren: ', len(filelist), '\n')

#   Header-Check erstes Spektrum
hdul = fits.open(filelist[0])
print('\nInfo zum ersten Spektrum der Liste:')
hdul.info()
print('\nKompletter Header des ersten Spektrums der Liste:')
print('\n', repr(fits.getheader(filelist[0], 0)))

# Leseroutine für wichtige Header-Daten der Spektrenliste
# sollte auf den konkreten Fall angepasst werden


def read_setup(filename):
    '''
    Parameter
    ----------
    filename : string
    Name des fitsfile

    Return
    -------
    Headerdaten nach Wunsch
    '''
    sp = fits.open(filename)
    header = sp[0].header
    sp.close()
# Hier anpassen, je nach gewünschten Headerdaten
# Falls von den nachfolgend bezeichneten Header-Keywords welche nicht
# vorhanden sind gibt es eine Fehlermeldung. Diese dann im return entfernen.
    return header['NAXIS1'], header['CRVAL1'], header['CDELT1'],\
           header['JD-OBS'], header['CUNIT1'], header['OBJECT'],\
           header['OBSERVER']


#   Print der mit der Leseroutine ermittelten Header-Daten für alle Spektren
print('\nAusgewählte Headerdaten in der Filelist:')
print('\n                        Pixel   start   step     Date        Einheit\
      Object      Beobachter\n')
print('Original:\n')
for f in filelist:
    print(f, read_setup(f))

ant = input('Möchten Sie an den Headerdaten etwas ändern? j/n : ')
# Headerkorrektur, Keywords und Inhalte an den jeweiligen Fall anzupassen
if ant == 'j':
    for f in range(len(filelist)):
        g = fits.open(filelist[f])
        g[0].header['OBJECT'] = 'omi Leo'
        g[0].header['OBSERVER'] = 'Mustermann'
        fits.update(filelist[f], g[0].data, g[0].header)
        g.close()


#   Header-Check aller Spektren nach der Korrektur
print('\nNach der Korrektur:\n')
for f in filelist:
    print(f, read_setup(f))
```

Figure 6.2: Programmlisting Header eines 1d-Spektrums (im fits-Format) anzeigen, ändern, aktualisieren.

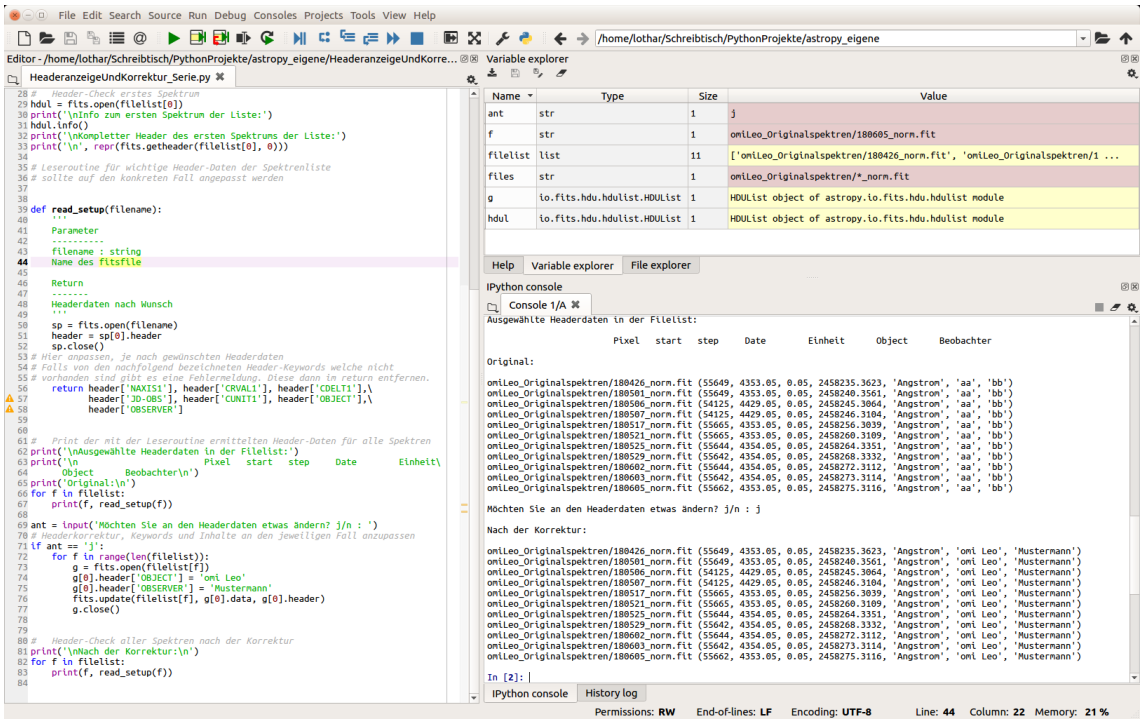


Figure 6.3: Bildschirmfoto nach Durchlauf des Programms in Spyder mit der Spektrenserie der Abb. 6.1.

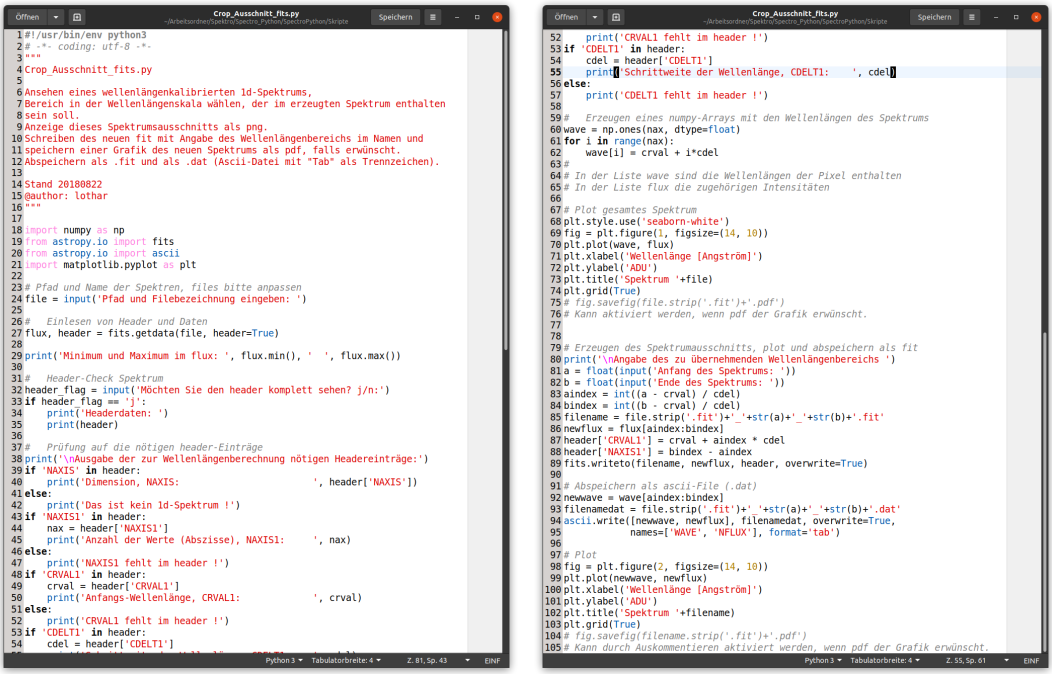


Figure 7.1: Programmlisting von „*Crop_ Ausschnitt_ fits.py*.

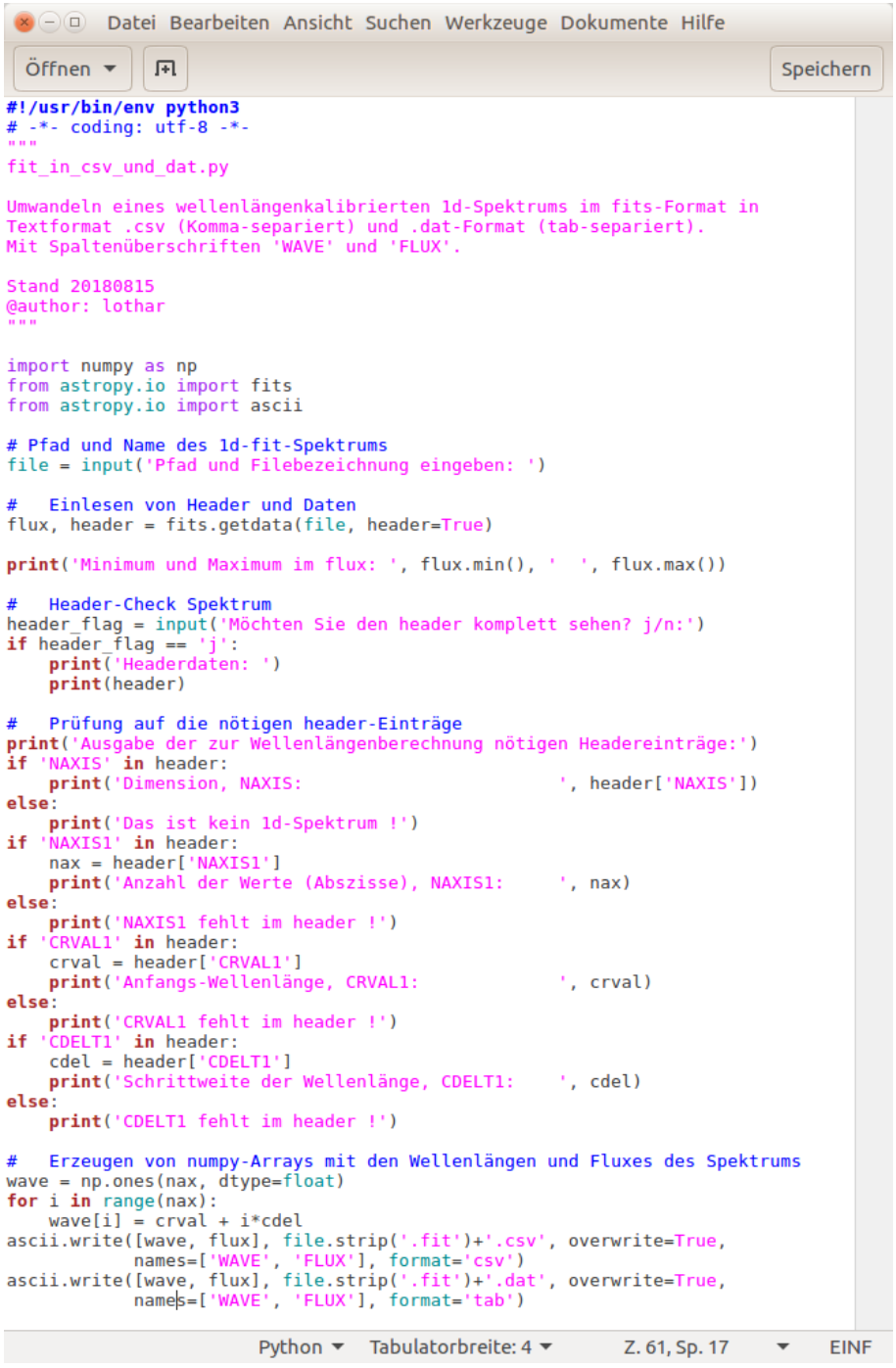


Figure 8.1: Skript fit_in_csv_und_dat.py zur Umwandlung von 1d-Spektren im fits-Format in Ascii-Dateien.

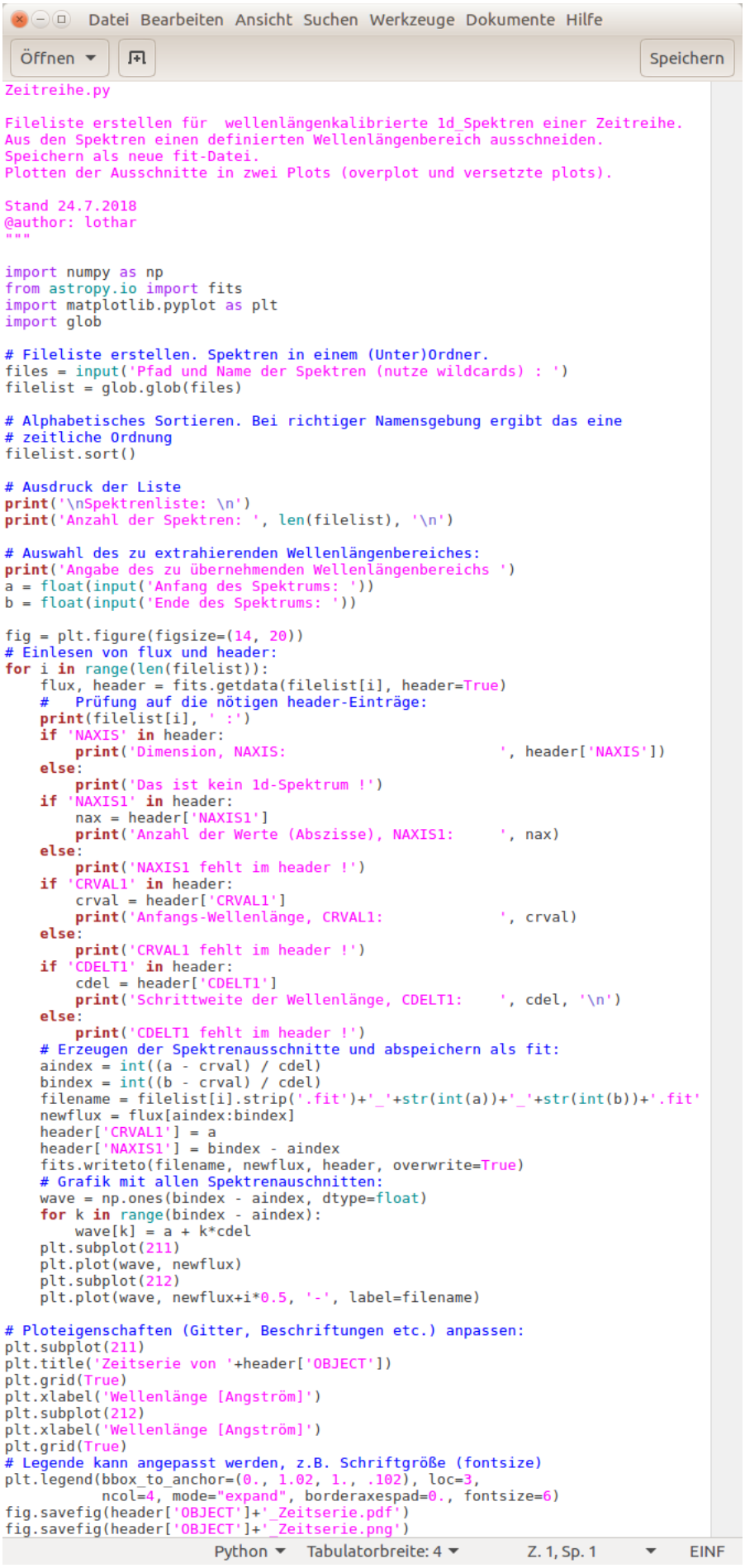


Figure 9.1: Skript zum Plotten von Zeitreihen von 1d-Spektren (.fits).

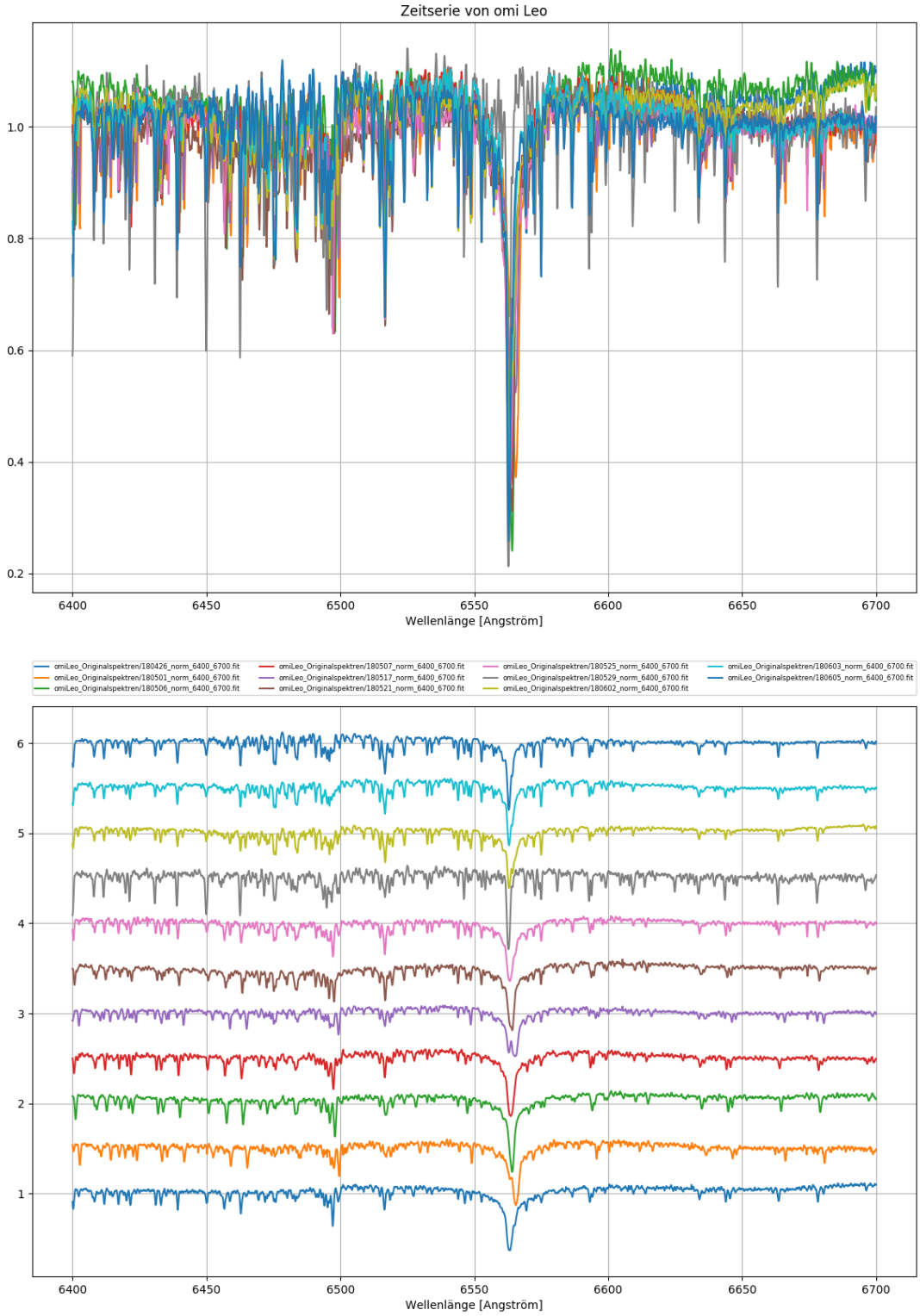


Figure 9.2: Plot einer Zeitreihe von 1d-Spektren. Oben alle übereinander geplottet, unten vertikal versetzt.

Part V.
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 verändern[24].

10. 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. 10.1 geplottet werden.

11. Plotten und erzeugen eines Ausschnitts aus einem ascii-1d-Spektrums

Ähnlich wie in Abschnitt 7 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. 11.1). Das Ergebnis des Skripts *PlotAndCrop_1d_ascii.py* sind eine Grafik des kompletten Spektrums und des erzeugten Ausschnitts (Abb. 11.2) sowie eine .dat-Datei des erzeugten Ausschnitts, entsprechend benahmt.

Part VI.
Konvolution (Faltung) von Modell-1d-Spektren mit Gaussfunktionen

Modellspektren (also theoretisch berechnete), wie man sie aus dem Internet beziehen kann[25], besitzen Dispersionen (Angström/Schrittweite), 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 Apparateprofil

[24] https://docs.python.org/3/library/csv.html#
[25] http://svo2.cab.inta-csic.es/theory/newov/
http://pollux.graal.univ-montp2.fr/
http://marcs.astro.uu.se/

Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente Hilfe

Öffnen 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

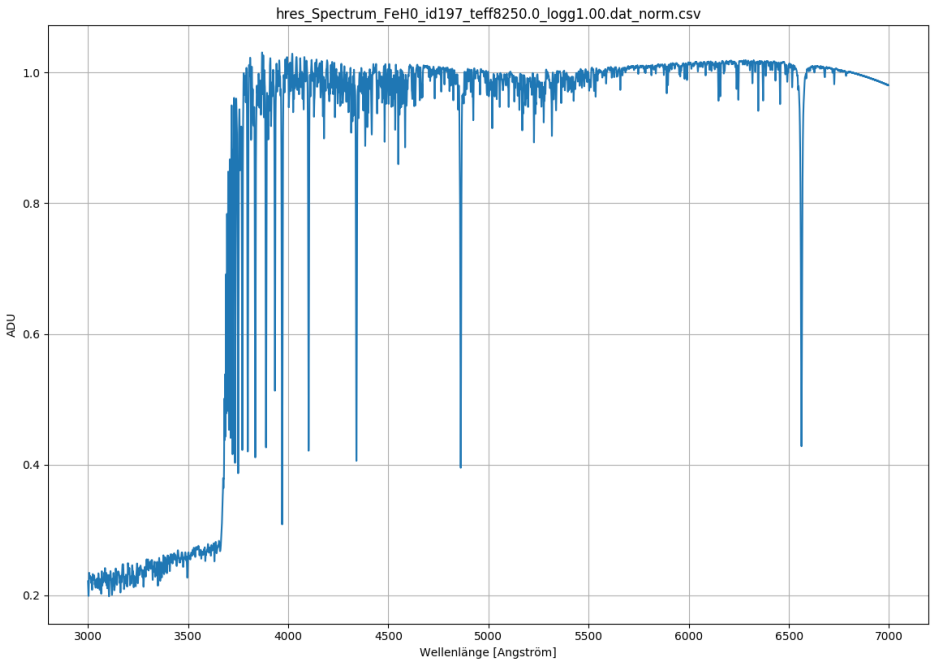


Figure 10.1: Listing von *1d_txt_plotten.py* und Abbildung eines ascii-1d-Spektrums erzeugt durch *1d_txt_plotten.py*.

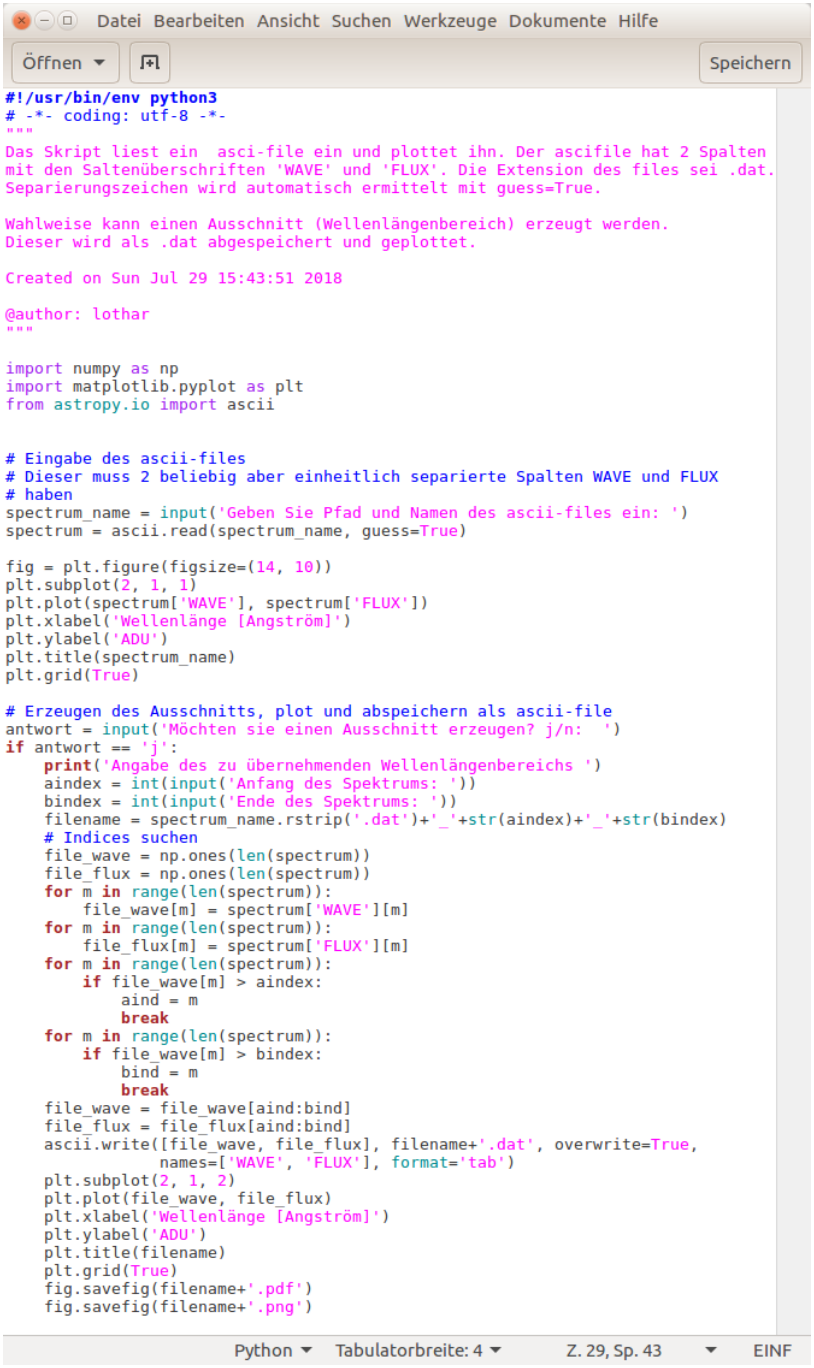


Figure 11.1: Plotten eines ascii-1d-Spektrums und erzeugen eines Ausschnitts

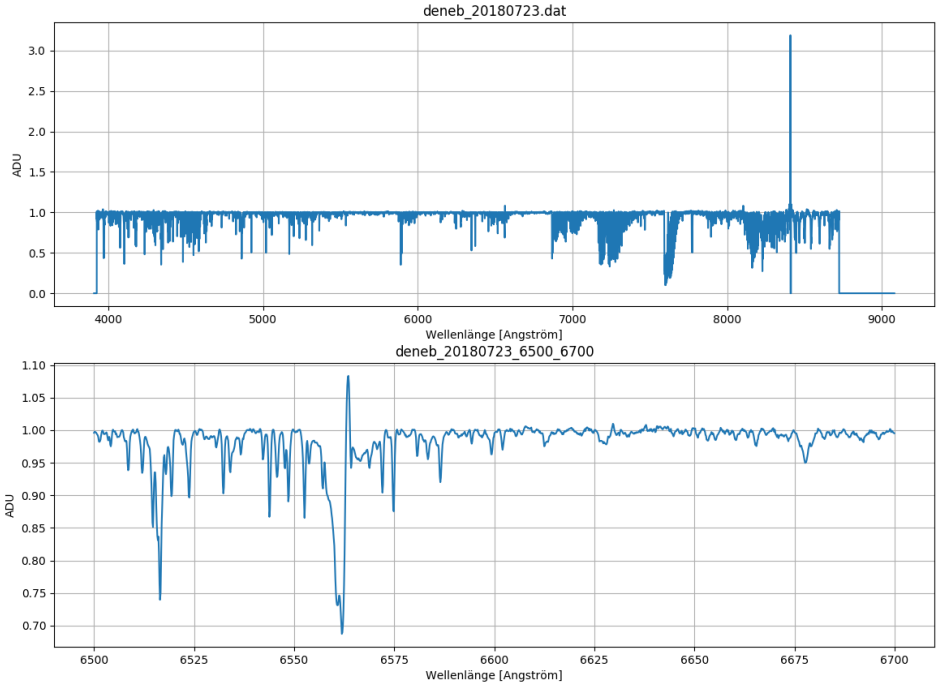


Figure 11.2: Resultierende Grafik des Skripts *PlotAndCrop_1d_ascii.py*. Oben das gesamte Echelle-Spektrum von Deneb, unten der erzeugte Ausschnitt von 6500 bis 6700 Angström.

der weniger aufgelösten zu falten (in Form der FWHM der schmalsten Linien im gemessenen Spektrum, am besten der terrestrischen Linien) .

12. *Pollux_Ausschnitt_convol.py*

Nachfolgend wird ein Python3-Skript *Pollux_Ausschnitt_convol.py* vorgestellt(Programmlisting in 12.1), welches Modellspektren aus der Pollux-Datenbank verarbeitet. Es wurde mit den physikalischen Parametern von Deneb (8400K, logg 3.8, F/H 1.0) 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, darunter die wichtigen Größen Startwellenlänge und step (Dispersion, Schrittweite) sowie eine yxz.spec-Datei, die 3 Spalten ohne Überschrift enthält: Wellenlänge, absoluter Flux und normierter Flux.

Nach dem Start des Programms werden nacheinander eingegeben: Pfad und Bezeichner des Spektrums (hier template_deneb.spec), Startwellenlänge lambda_min im Spektrum entsprechend der Datei template_deneb.t
Schrittweite (deltalambda) und die gewünschte Standardabweichung für die Faltung in ganzen Vielfachen der Schrittweite (hier wurde 30 gewählt). Dann wird noch angegeben, für welchen Wellenlängenbereich die Faltung durchgeführt werden soll (hier 6400 bis 6700 Angström). Es wird dann eine Grafik ausgegeben mit dem gesamten Spektrum, darunter der gewählte Ausschnitt und ganz unten als dritter Plot der gewählte Ausschnitt nach der Faltung. Abgespeichert werden die Dateien template_deneb_convolved.dat und die Grafik im PNG und im PDF-Format. In 12.2 ist die Grafik für das hier durchgeführte Beispiel gezeigt.

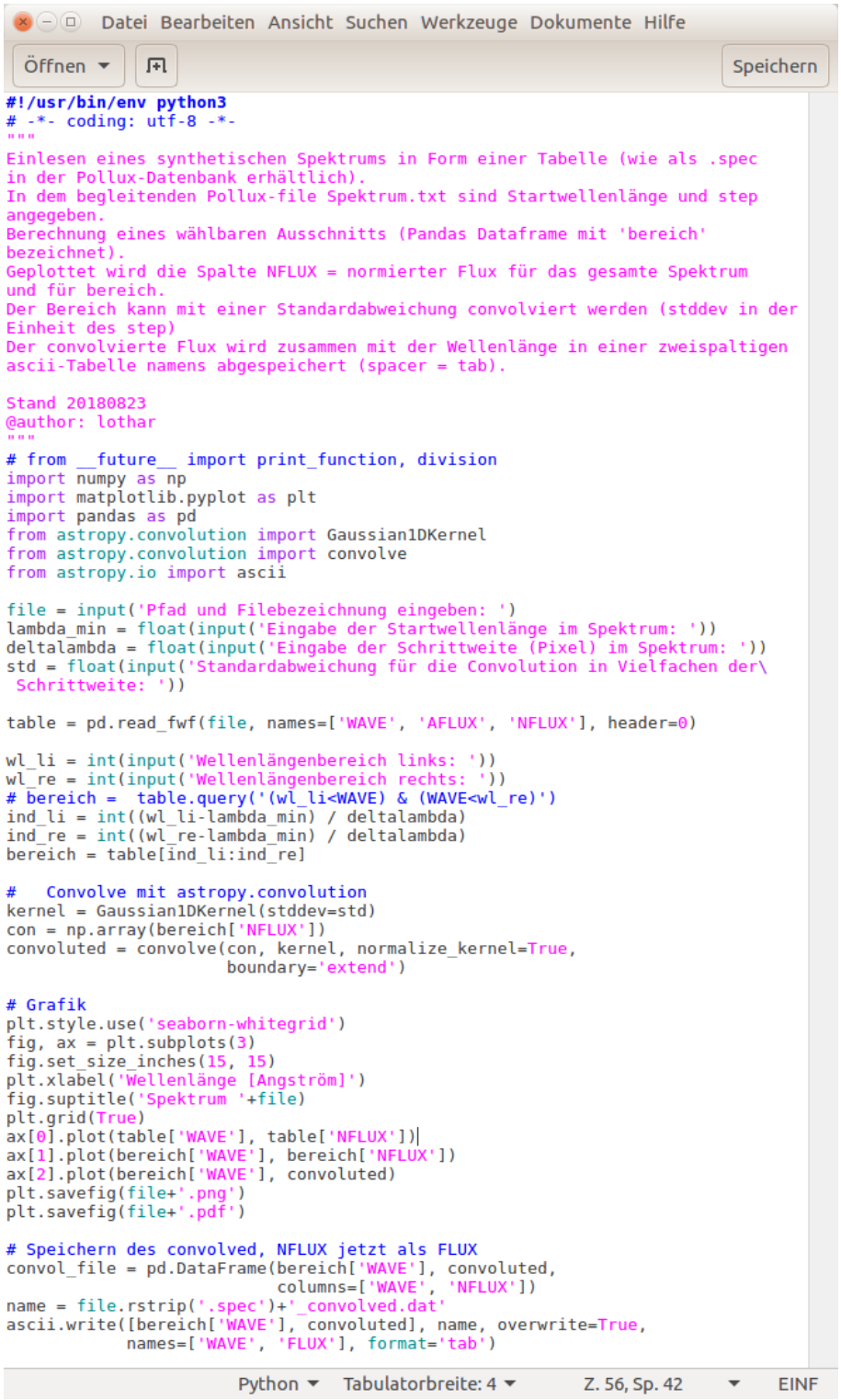


Figure 12.1: Programmlisting des Skripts *Pollux_ Ausschnitt_ convol.py*.

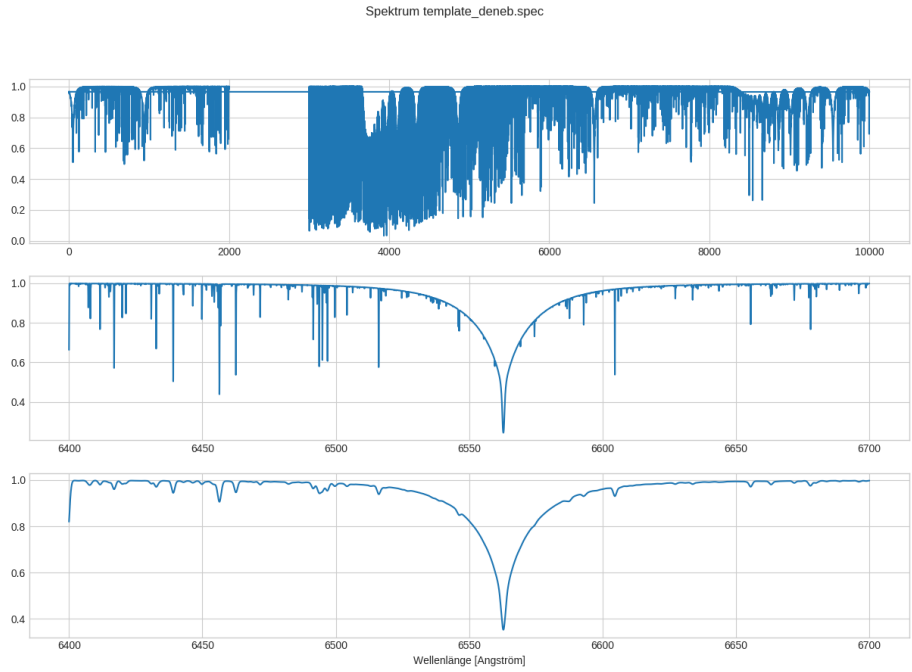


Figure 12.2: Grafik des Skripts *Pollux_ Ausschnitt_ convol.py*, oben das gesamte Spektrum, mittig der gewählte Ausschnitt und unten nach Faltung.

13. *convol_ dat.py*

Ein einfaches Python3-Skript zum Falten eines 1d-Spektrums mit einer Gaussfunktion, das in Form einer 2-spaltigen ascii-Datei vorliegt[26], ist nachfolgend aufgeführt. Eingegeben wird der Pfad/Name des ascii-Files und der Standardabweichung der Gaussfunktion, mit der gefaltet werden soll (in Vielfachen 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.

14. Vergleich von theoretischem und gemessenen Spektrum (overplot)

Ist das nach vorstehendem Abschnitt 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. 14.1 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. 14.2.

Part VII.
Normierung von 1d-Spektren

Das Normieren von 1d-Spektren auf das Kontinuum ist eine häufig zu erledigende Aufgabe. Ein Python-Skript, welches eine ganze (Zeit)Serie von 1d-Spektren im fits-Format *automatisch* auf das Kontinuum

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

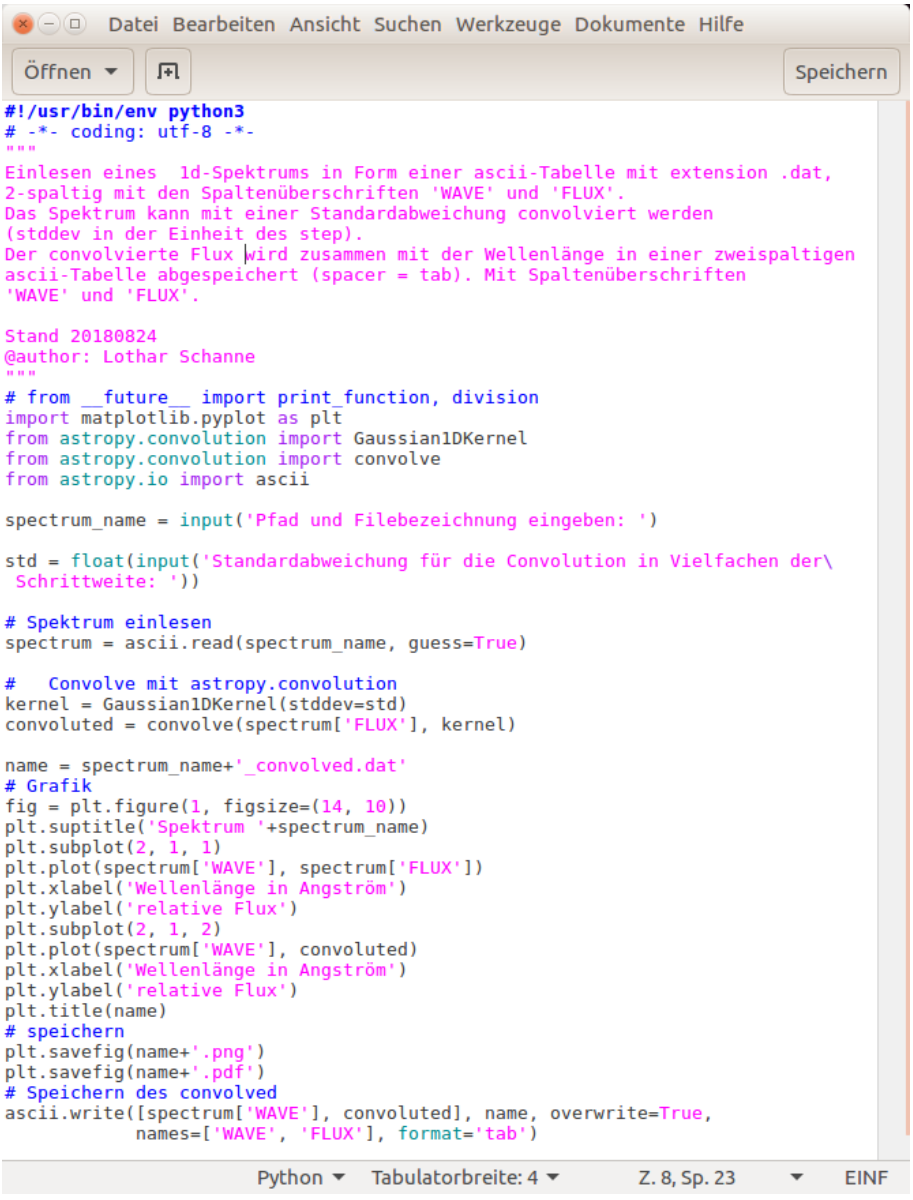


Figure 13.1: Das Skript *convol_ dat.py* zum Falten eines Spektrums mit einer beliebigen Gaussfunktion.

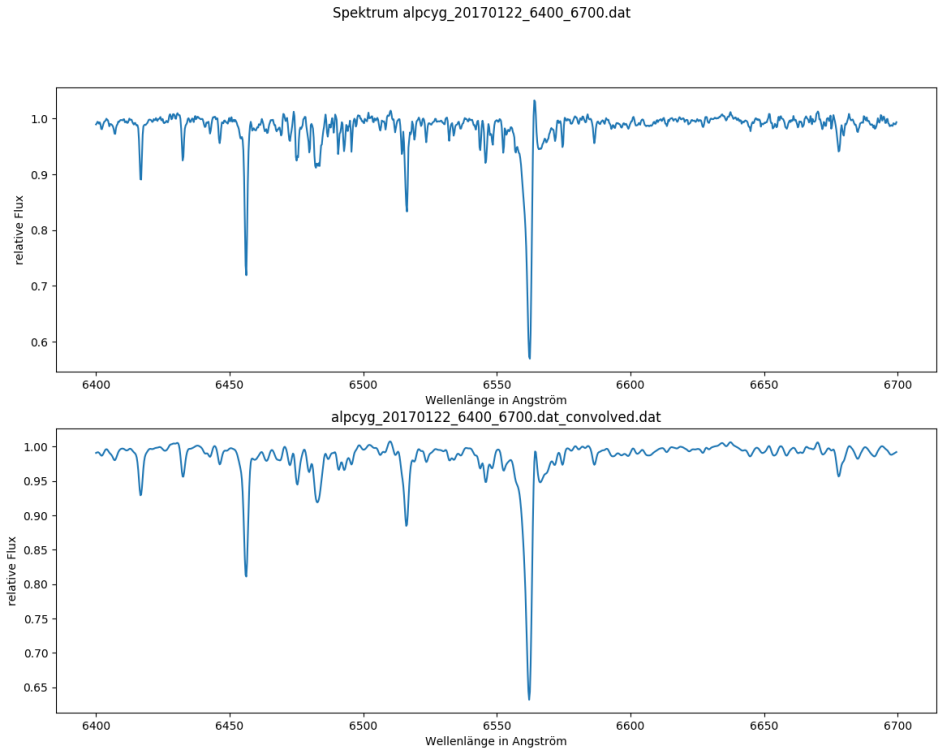


Figure 13.2: Ergebnisgrafik des Skripts *convol_ dat.py* zum Falten eines Spektrums mit einer beliebigen Gaussfunktion. Hier wurde mit dem Parameter $std = 2$ (≈Verdoppelung des gerade noch aufgelösten Elements, Halbierung der relativen Auflösung R) gearbeitet. Die Breite der Linien wurde dadurch etwa verdoppelt.

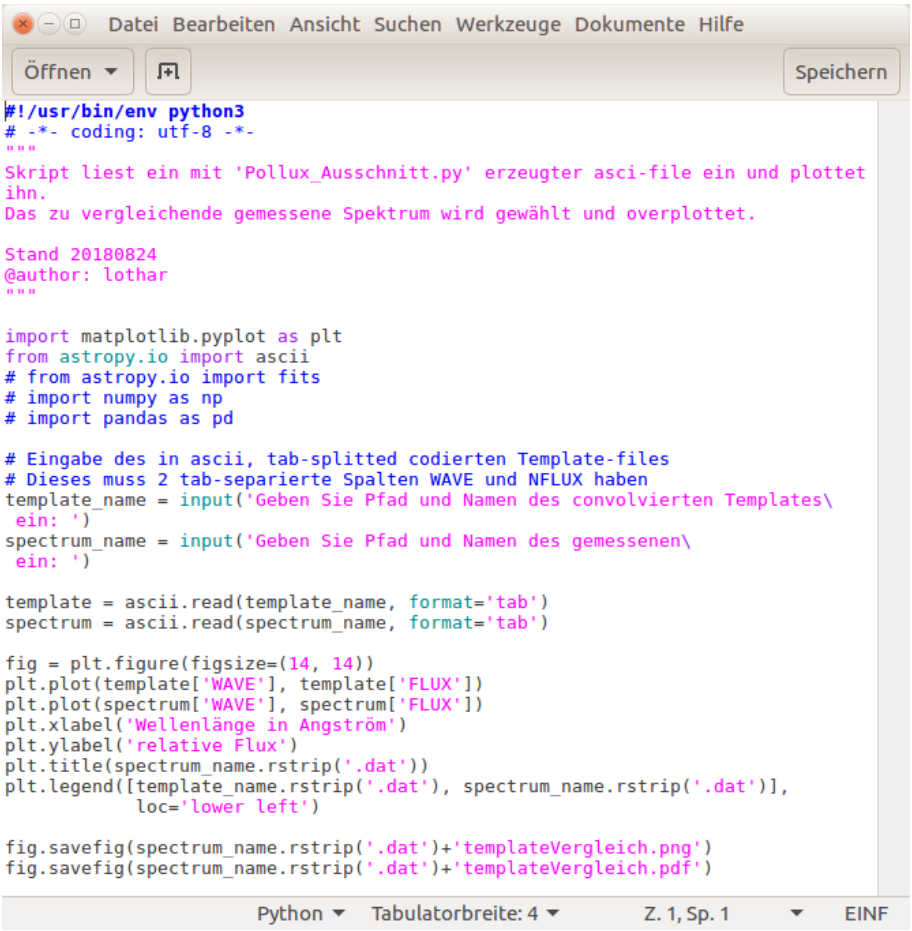


Figure 14.1: Skript-Listing von Vergleichsplot_Modell_Messung.py.

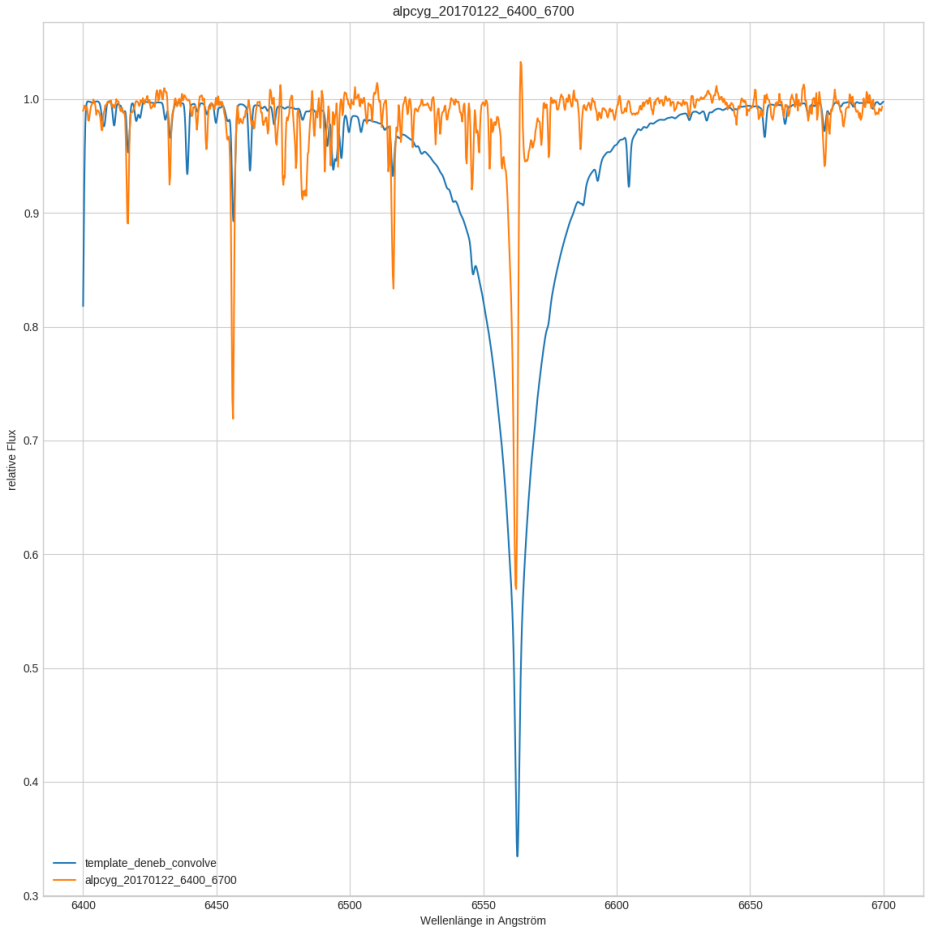


Figure 14.2: Von *Vergleichsplot_ Modell_ Messung.py* erzeugte Grafik.

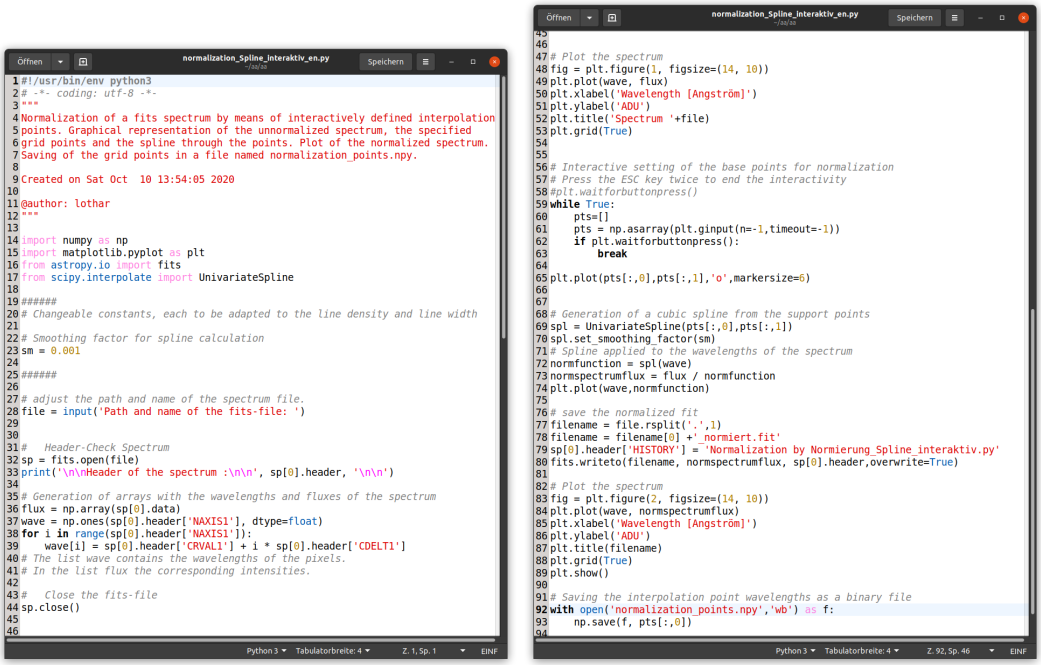


Figure 14.3: Listing des Python-Programms zur interaktiven Normierung eines 1d-Spektrums im fits-Format

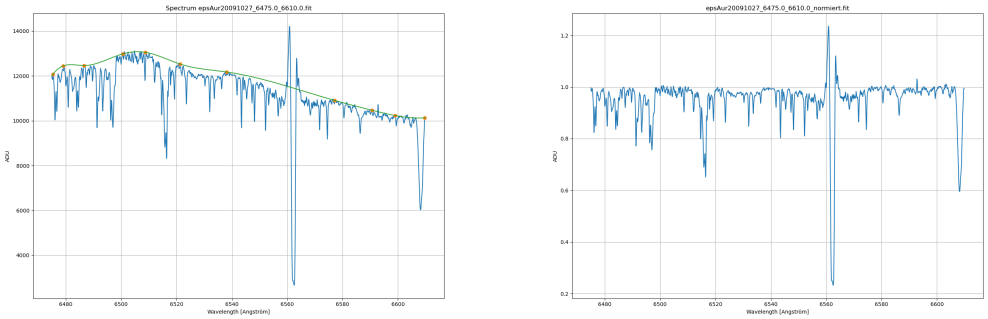


Figure 14.4: Beispiel einer interaktiven Normierung. Links das zu normierende Spektrum mit den Normierungs-Stützstellen und dem Spline als Normierungsfunktion. Rechts das normierte Ergebnis-Spektrum.

normieren kann, ist unter dem link *https://lotharschanne.wordpress.com/automatic-normalization-of-spectra-and-time-series-of-spectra-in-fits-format/* ladbar. Unter dem link ist auch eine detaillierte Beschreibung des Skripts und seiner Verwendung (in Englisch) zu finden.

Nachfolgend wird ein einfacheres Skript besprochen, das es erlaubt, ein 1d-Spektrum im fits-Format interaktiv zu normieren (Code in 14.3). Zur Normierung müssen interaktiv durch Anklicken Punkte, die das Kontinuum repräsentieren, ausgewählt werden. Durch diese wird dann ein kubischer Spline gelegt, dessen Steifheit mit einem „Smoothing-Faktor“ *sm* gesteuert wird (in Zeile 23 festzulegen, sm = 1 bedeutet, dass alle festgelegten Punkte vom Spline berührt werden, sm = 0 dagegen bedeutet eine Ausgleichsgerade durch die ausgewählten Stützpunkte). In einer Grafik werden das zu normierende Spektrum, die ausgewählten Stützpunkte und der Spline gezeigt, in einem zweiten Grafikfenster ist das normierte Spektrum zu sehen, das auch als fits-Datei abgespeichert wird. Zudem werden die Stützpunkte in einer Binärdatei gespeichert.

Part VIII.

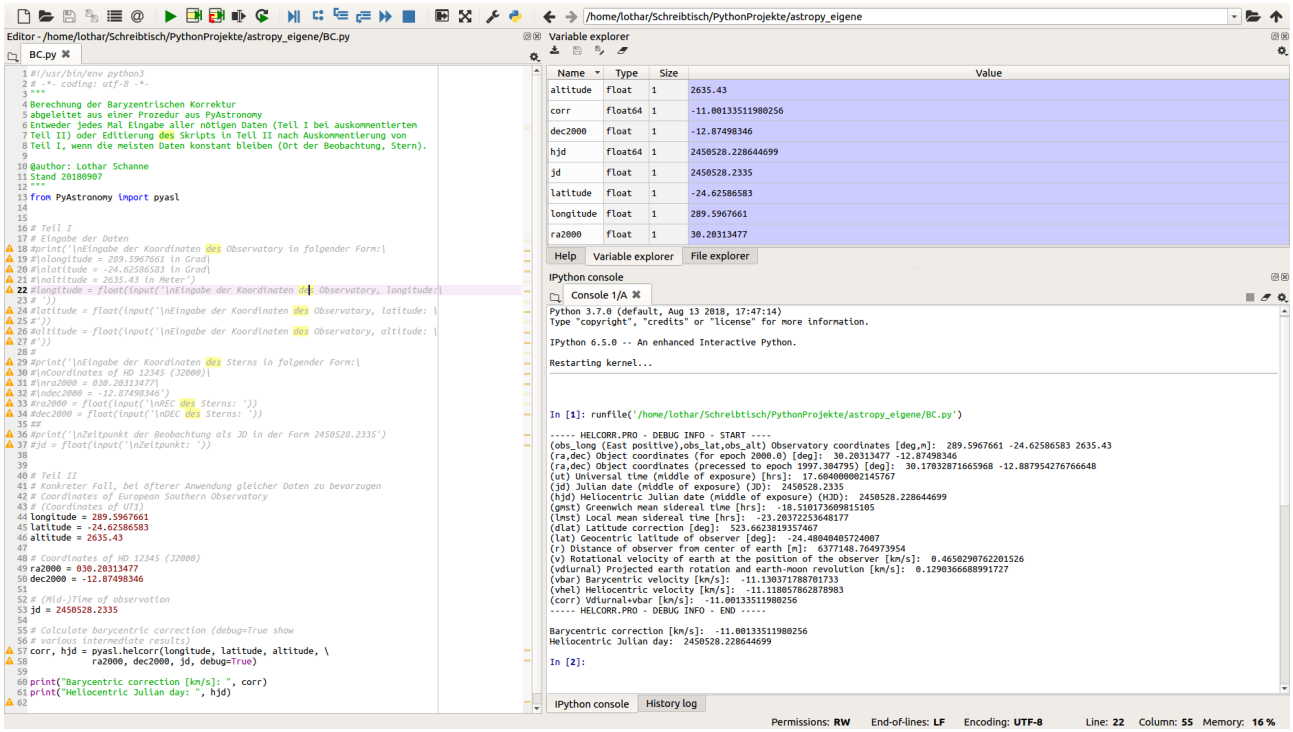


Figure 15.1: Berechnung der Baryzentrischen Korrektur mit dem Skript BC.py.

Baryzentrische Korrektur (barycentric correction BC)

15. 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 (Teil I vorher auskommentieren, wie in Abb. 15.1 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.

16. BC-Korrektur einer Serie von 1d-Spektren 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. 16.1 gelistet.

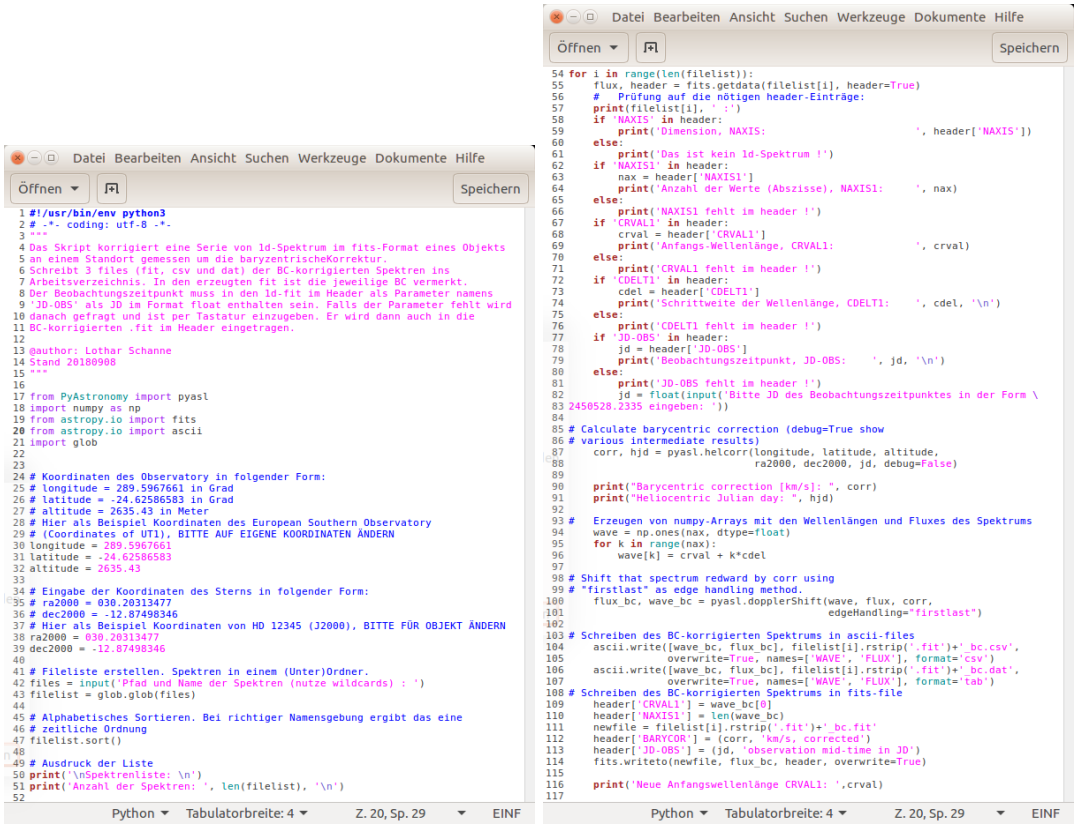


Figure 16.1: Programmlisting von *BC_ Dopplershift_ 1dSpectrum_ Serie.py*.

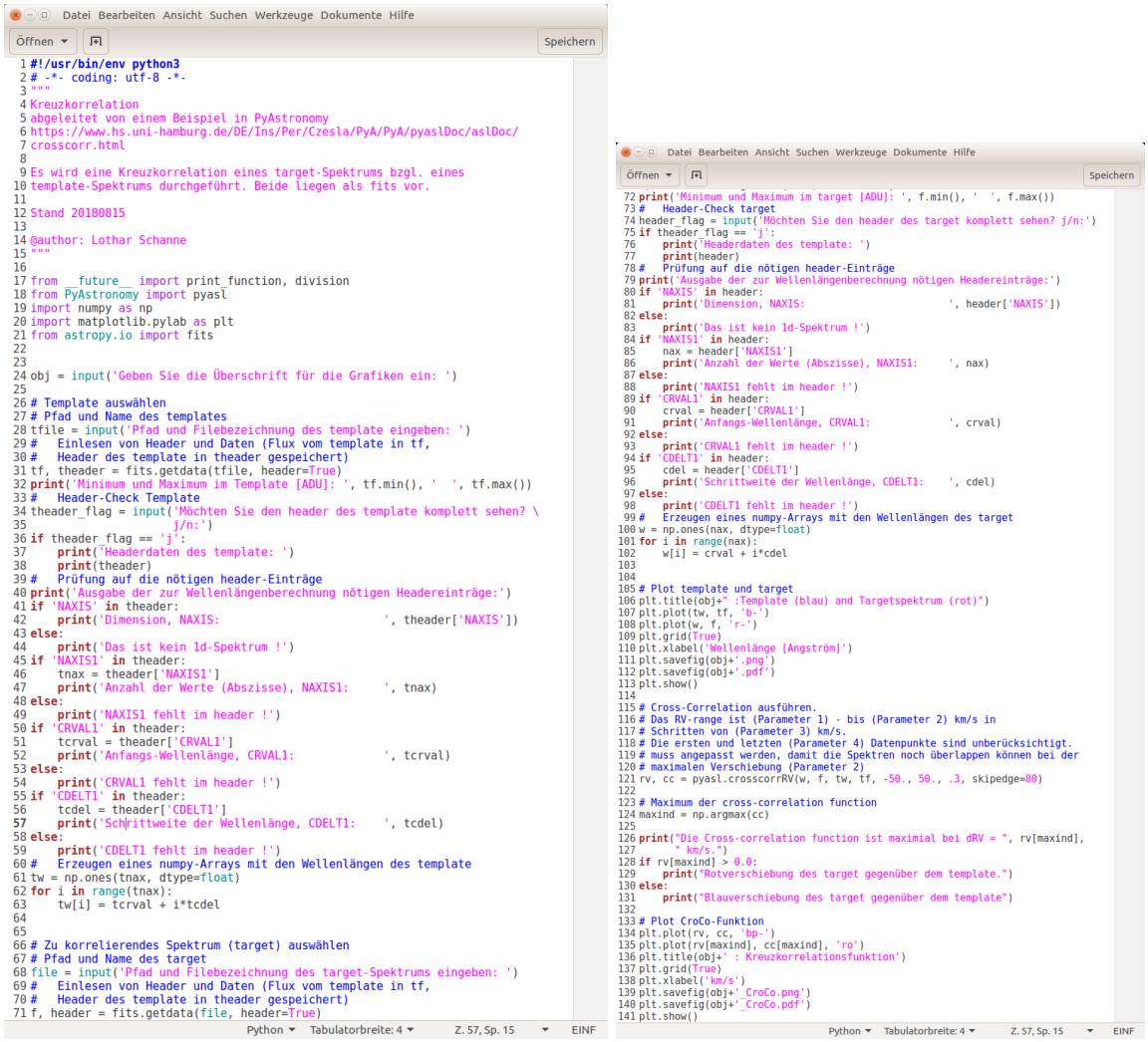
Es benötigt neben den geografischen Koordinaten für den Beobachtungsstandort und die Himmelskoordinaten des Objekts den Beobachtungszeitpunkt, und zwar im Header-Parameter 'JD-OBS' als Gleitkommazahl (float). Falls der Beobachtungszeitpunkt in anderer Form vorliegt wird danach gefragt und er kann über die Tastatur für jedes Spektrum eingegeben werden. Das Programm speichert die baryzentrisch korrigierten Spektren in drei Formaten ab: fit, dat und csv. Die Dateien werden neu benahmt, mit der Erweiterung _bc am Ende ihres ursprünglichen Bezeichners. Im fit ist der Header um die BC erweitert (und im Falle der manuellen Eingabe des Beobachtungszeitpunktes wird auch der neue Header-Parameter 'JD-OBS' angelegt).

Part IX.
Kreuzkorrelationen (KK)

17. 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" 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 nicht[27], 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

[27] z.B. terrestrische Linien sind in einem theoretischen, berechneten Spektrum nicht enthalten



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Kreuzkorrelation
abgeleitet von einem Beispiel in PyAstronomy
https://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyaslDoc/aslDoc/
crosscorr.html

Es wird eine Kreuzkorrelation eines target-Spektrums bzgl. eines
template-Spektrums durchgeführt. Beide liegen als fits vor.

Stand 20180815

@author: Lothar Schanne
"""

from __future__ import print_function, division
from PyAstronomy import pyasl
import numpy as np
import matplotlib.pylab as plt
from astropy.io import fits


obj = input('Geben Sie die Überschrift für die Grafiken ein: ')

# Template auswählen
# Pfad und Name des templates
tfile = input('Pfad und Filebezeichnung des template eingeben: ')
#   Einlesen von Header und Daten (Flux vom template in tf,
#   Header des template in theader gespeichert)
tf, theader = fits.getdata(tfile, header=True)
print('Minimum und Maximum im Template [ADU]: ', tf.min(), '  ', tf.max())
#   Header-Check Template
theader_flag = input('Möchten Sie den header des template komplett sehen? \
                     j/n:')
if theader_flag == 'j':
    print('Headerdaten des template: ')
    print(theader)
#   Prüfung auf die nötigen header-Einträge
print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
if 'NAXIS' in theader:
    print('Dimension, NAXIS:                          ', theader['NAXIS'])
else:
    print('Das ist kein 1d-Spektrum !')
if 'NAXIS1' in theader:
    tnax = theader['NAXIS1']
    print('Anzahl der Werte (Abszisse), NAXIS1:     ', tnax)
else:
    print('NAXIS1 fehlt im header !')
if 'CRVAL1' in theader:
    tcrval = theader['CRVAL1']
    print('Anfangs-Wellenlänge, CRVAL1:             ', tcrval)
else:
    print('CRVAL1 fehlt im header !')
if 'CDELT1' in theader:
    tcdel = theader['CDELT1']
    print('Schrittweite der Wellenlänge, CDELT1:    ', tcdel)
else:
    print('CDELT1 fehlt im header !')
#   Erzeugen eines numpy-Arrays mit den Wellenlängen des template
tw = np.ones(tnax, dtype=float)
for i in range(tnax):
    tw[i] = tcrval + i*tcdel


# Zu korrelierendes Spektrum (target) auswählen
# Pfad und Name des target
file = input('Pfad und Filebezeichnung des target-Spektrums eingeben: ')
#   Einlesen von Header und Daten (Flux vom template in tf,
#   Header des template in theader gespeichert)
f, header = fits.getdata(file, header=True)
print('Minimum und Maximum im target [ADU]: ', f.min(), '  ', f.max())
#   Header-Check target
header_flag = input('Möchten Sie den header des target komplett sehen? j/n:')
if theader_flag == 'j':
    print('Headerdaten des template: ')
    print(header)
#   Prüfung auf die nötigen header-Einträge
print('Ausgabe der zur Wellenlängenberechnung nötigen Headereinträge:')
if 'NAXIS' in header:
    print('Dimension, NAXIS:                          ', header['NAXIS'])
else:
    print('Das ist kein 1d-Spektrum !')
if 'NAXIS1' in header:
    nax = header['NAXIS1']
    print('Anzahl der Werte (Abszisse), NAXIS1:     ', nax)
else:
    print('NAXIS1 fehlt im header !')
if 'CRVAL1' in header:
    crval = header['CRVAL1']
    print('Anfangs-Wellenlänge, CRVAL1:             ', crval)
else:
    print('CRVAL1 fehlt im header !')
if 'CDELT1' in header:
    cdel = header['CDELT1']
    print('Schrittweite der Wellenlänge, CDELT1:    ', cdel)
else:
    print('CDELT1 fehlt im header !')
#   Erzeugen eines numpy-Arrays mit den Wellenlängen des target
w = np.ones(nax, dtype=float)
for i in range(nax):
    w[i] = crval + i*cdel


# Plot template und target
plt.title(obj+" :Template (blau) and Targetspektrum (rot)")
plt.plot(tw, tf, 'b-')
plt.plot(w, f, 'r-')
plt.grid(True)
plt.xlabel('Wellenlänge [Angström]')
plt.savefig(obj+'.png')
plt.savefig(obj+'.pdf')
plt.show()

# Cross-Correlation ausführen.
# Das RV-range ist (Parameter 1) - bis (Parameter 2) km/s in
# Schritten von (Parameter 3) km/s.
# Die ersten und letzten (Parameter 4) Datenpunkte sind unberücksichtigt.
# muss angepasst werden, damit die Spektren noch überlappen können bei der
# maximalen Verschiebung (Parameter 2)
rv, cc = pyasl.crosscorrRV(w, f, tw, tf, -50., 50., .3, skipedge=80)

# Maximum der cross-correlation function
maxind = np.argmax(cc)

print("Die Cross-correlation function ist maximial bei dRV = ", rv[maxind],
      " km/s.")
if rv[maxind] > 0.0:
    print("Rotverschiebung des target gegenüber dem template.")
else:
    print("Blauverschiebung des target gegenüber dem template")

# Plot CroCo-Funktion
plt.plot(rv, cc, 'bp-')
plt.plot(rv[maxind], cc[maxind], 'ro')
plt.title(obj+' : Kreuzkorrelationsfunktion')
plt.grid(True)
plt.xlabel('km/s')
plt.savefig(obj+'_CroCo.png')
plt.savefig(obj+'_CroCo.pdf')
plt.show()
```

Figure 17.1: Kreuzkorrelationsskript für 1d-fits-Spektren.

nutzt man besser Spektrumausschnitte, 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 vorhergehende baryzentrische Korrektur nach VIII für alle beteiligten Spektren erforderlich.

In 17.1 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 ausgegeben[28] 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 in 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

[28] Zur 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.

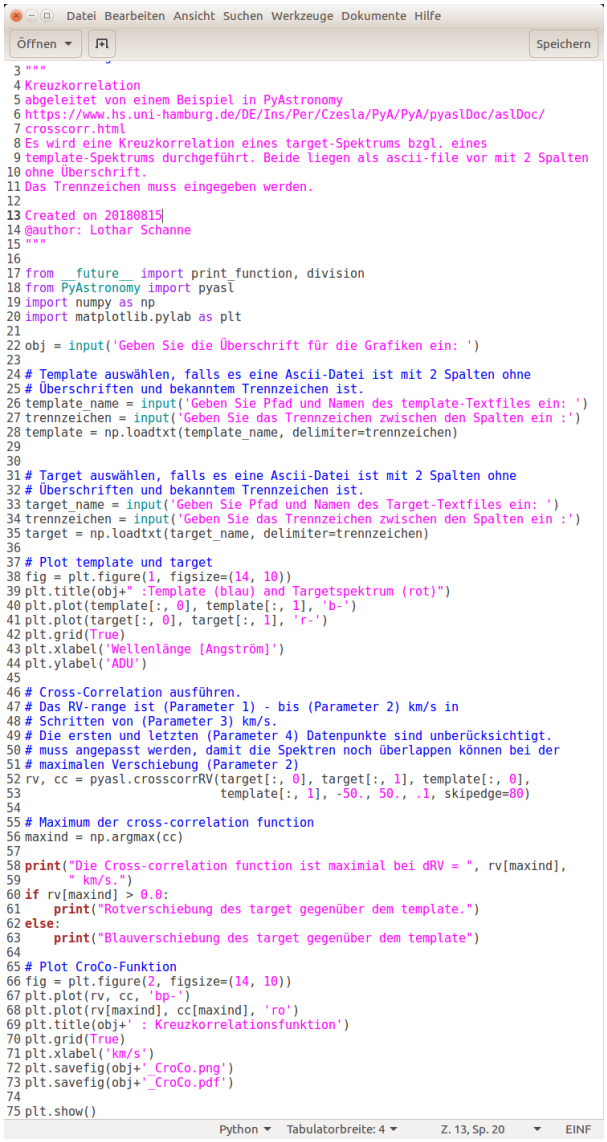


Figure 18.1: Kreuzkorrelationsskript für 1d-ascii-Spektren.

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.

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

Analog zu 17.1 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.