

Spectro-Python

Auswertung optischer astronomischer Spektren mit Hilfe von Python 3

von Dr. Lothar Schanne

Stand 13.12.2024

Contents

I. Einleitung	5
II. Motivation	6
III. Installation einer persönlichen Python-Version	7
1. Allgemeines	7
2. Installation einer Python-Version im eigenen User-Verzeichnis	8
2.1. Anaconda	8
2.2. Python-Environment	8
2.3. Vollständige unabhängige Python-Version im Userhomeverzeichnis	9
3. Installation einer Entwicklungsumgebung (Spyder) und SciPy (wissenschaftliches Rechnen)	10
4. Verwendung von Spyder	11
5. Spektroskopiespezifische Python-Bibliotheken	11
6. Zur Einführung ein erstes spektroskopisches Programm	12
IV. Beschreibung der einzelnen Skripte	17
7. Ordner ascii	17
7.1. 1d_dat_read_plot_csv_speichern.py	17
7.2. 1d_txt_plotten.py	19

7.3.	comparison_plot_2spectra_ascii.py	19
7.4.	PlotAndCrop_1d_ascii.py	19
8.	Ordner astroplanner	19
8.1.	astroplanner.py	19
8.2.	astroplanner_fuer_Objektliste.py	19
9.	Ordner BC	20
9.1.	BC_1d_spectra_fits_series.py	20
10.	Ordner binaries (Periodenbestimmung und Berechnung von Orbitalelementen)	20
10.1.	Berechnung_Phasenplot_mitBekannterPeriode	21
10.2.	PDM_Periodenanalyse.py	21
10.3.	Periodenanalyse.py	21
10.4.	Phasenberechnung.py	21
10.5.	Phasenberechnung_ausSpektren_mitvorgegebenenToUndPeriode.py	22
10.6.	Orbitalelemente_ausJDundGemessenenRV_optimieren.py	22
10.7.	RV_Berechnung_ausOrbitalelementenUndJD.py	22
11.	Ordner Convolution	22
11.1.	convol_dat.py	22
11.2.	convol_fits.py	23
11.3.	Pollux_Ausschnitt_convolver.py	23
11.4.	rotationsverbreiterung_fits.py	23
12.	Ordner DarstellungImGeschwindigkeitsraum	23
12.1.	fitsSerie_Linie_imGeschwindigkeitsraum	23
12.2.	fitsSerie_Linie_imGeschwindigkeitsraum_JDBereich.py	23
12.3.	fitsSerie_Linien_imGeschwindigkeitsraum_JDBereich.py	24
13.	Ordner EW	24
13.1.	Zeitreihe_EW.py	24
13.2.	Zeitreihe_EW_Linientiefe_Linienwellenlaenge.py	24
13.3.	Zeitreihe_EW_grafisch_mehrereBereicheInEinerLinie.py	25
13.4.	Zeitreihe_EWs_mehrereBereicheInEinerLinie.py	25
14.	Ordner fits	25
14.1.	1d_fitSpektrum_ansehen.py	25
14.2.	1d_fitSpektrum_ansehen_mitLinienidentifikation.py	26
14.3.	1d_fitSpektrum_ansehen_mitLinienidentifikationPerElement.py	26
14.4.	1d_fitSpektrum_ansehen_Mit_Subplots.py	26
15.	Ordner Formatumwandlungen	26
15.1.	Formatumwandlungen.py	26

16. Ordner KK (Kreuzkorrelationen)	26
16.1. CrossCorrelation_dat_Serie.py	27
16.1.1. CrossCorrelation_fits_Serie.py	27
16.2. CrossCorrelation_fits_Serie_einzelneLinien.py	27
17. Ordner Lichtkurven	27
17.1. AAVSO.py	27
18. Ordner Linienfitting	27
18.1. FitLinie.py	28
18.2. FitLinie_automatischParametrisiert.py	28
18.3. FitLinie_PositiverUndNegativerGauss.py	29
18.4. FitLinie_PositiverUndNegativerVoigt.py	29
19. Ordner Normierung	30
19.1. automaticNormalization_timeseries_20240115.py	30
19.2. 1Punkt_Normierung.py	30
19.3. Normierung_Abstandsmethode_Splinemethode_20241126.py	30
19.4. Normierung_einzelne Linien_20241105.py	31
19.5. Normierung_findPeaks_Polynommethode_20241111.py	31
19.6. Normierung_findPeaks_Splinemethode_20241111.py	31
19.7. Normierung_Intervalle_Splinemethode_20241127.py	31
19.8. Normierung_Steigung_Krümmung_Splinemethode_20241126.py	32
20. Ordner NRES	32
20.1. NRES_auslesen.py	33
20.2. VacToAir.py	33
20.3. PlotAllerOrdnungen.py	34
20.4. heliocentricCorrection_NRES.py	34
21. Ordner Rebinning_gemitteltesSpektrum_Differenzspektren	34
21.1. newbinning_mittleresSpektrum.py	34
21.2. Differenzspektren.py	34
21.3. Differenzspektren_mitPlotAlle.py	35
22. Ordner RV_Korrektur	35
22.1. RVKorrektur_1d_spectra_fits_series.py	35
22.2. RVKorrekturPerTemplate_1d_spectra_fits_series.py	35
22.3. terrLinien_RVKorrektur_Regression_1d_spectra_fits_series	35
22.4. terrLinien_RVKorrektur_RBF_1d_spectra_fits_series.py	36
23. Ordner RV_Messung	36
23.1. RV_MessungAnLinie_fits_interaktiv.py	36
23.2. RV_MessungAnLinie_Zeitserie_dat_perInteraktion.py	37
23.3. RV_MessungAnLinie_Zeitserie_dat_perInteraktion_regression_2Linien.py 37	

23.4. RV_MessungAnLinie_Zeitserie_dat_perRegression.py	37
23.5. RV_MessungAnLinie_Zeitserie_fits_perGaussfit.py	37
23.6. RV_MessungAnLinie_Zeitserie_fits_perRBF.py	37
23.7. RV_MessungAnLinie_Zeitserie_fits_perRegression.py	38
23.8. RV_MessungAnLinie_Zeitserie_fits_perRegression_AbsUndEmi.py	38
23.9. RV_MessungAnLinie_Zeitserie_fits_perSpline.py	38
24. Ordner SNR	38
24.1. SNR_betasigma.py	38
24.2. SNR_fits_mehrereBereiche.py	39
24.3. verrauschen.py	39
25. Ordner theoretische Spektren	39
25.1. ascii_Spektrum_Ausschnitt_rebinned_convolve	39
25.2. KunstspektrumErzeugen.py	39
25.3. Kurucz_Modell_GaussBroadened.py	40
25.4. Pollux_Ausschnitt_rebinned_convolve_spec.py	40
26. Ordner Serien	40
26.1. AnimationAusPlots.py	40
26.2. ascii_timeseries_einPlot_mitOffset	40
26.3. badPixFilter.py	41
26.4. badPixFilter_VergleichNachbarpixel.py	41
26.5. Crop_Ausschnitt_Zeitserie_fits.py	41
26.6. Crop_Ausschnitt_Zeitserie_fits.py	41
26.7. DynamischerSpektrenplot	41
26.8. FilesMitGeringerDispersionLoeschen_fits.py	41
26.9. fit_Serie_in_csv_und_dat.py	41
26.10 HeaderanzeigeUndKorrektur_Serie.py	42
26.11 Headerprint_Serie_csv.py	42
26.12 JD_EintragInHeader.py	42
26.13 JD_InDateiname_uebernehmen.py	42
26.14 JD_und_Anfangswellenlaenge_InDateiname_uebernehmen.py	42
26.15 MittelungVon_fits.py	42
26.16 timeseries_einPlot_mitOffset.py	42
26.17 timeseries_einPlot_mitOffset_JDBereich.py	42
26.18 timeseries_einPlot_mitOffset_JD_Bereich_Wellenlangenbereich	43
26.19 timeseries_einPlot_mitOffset_Wellenlangenbereich	43
26.20 timeseries_minUndmax.py	43
26.21 timeseries_plot_JeEineGrafik.py	43
26.22 Ueberpruefung_JD_imHeader.py	43
26.23 Wellenlaengenbereich_Zeitserie_fits	43
26.24 Zeitserie_Beobachtungszeitpunkte.py	43
26.25 Zeitserie_Spektren_smoothen.py	44

Part I.

Einleitung

Diese Sammlung von Pythonskripten ist in den letzten Jahren entstanden, in denen ich immer wieder - teilweise umfangreiche - astronomische Spektrenserien von einem Himmelsobjekt im optischen Wellenlängenbereich auswerten wollte, aber immer wieder feststellte, dass die üblichen Spektrenbearbeitungsprogramme keine Zeitserien oder sonstige Serien von 1d-Spektren bearbeiten können. Deshalb liegt der Fokus dieser selbst geschriebenen Pythonskripte auf der Auswertung von Serien von 1d-Spektren, meist im fits-Format. Wobei die Skripte natürlich auch auf einzelne Spektren angewendet werden können. Dabei sind alle Auswertemethoden vorhanden, die mir in meiner Praxis vorgekommen sind. Angefangen mit der Normierung auf das Kontinuum, heliozentrischen Korrektur, Bestimmung von Äquivalentweiten von einzelnen Linien, Bestimmung von Radialgeschwindigkeiten durch Kreuzkorrelation oder aus der Doppler-Verschiebung einzelner Linien, Berechnung theoretischer Spektren und die Anpassung ihrer Auflösung an die des eigenen Spektrographen, grafische Darstellung von Spektren etc. Wie gesagt, nicht nur für einzelne Spektren sondern für eine Vielzahl von Spektren (z.B. Zeitserien eines Objektes). Innerhalb der Skripte sind viele Kommentare untergebracht, so dass auch mit rudimentären Pythonkenntnissen die Arbeitsweise eines Skripts verstanden wird oder auch die fallweise notwendige Anpassung der Skripte an den eigenen Bedarf (z.B. die Eintragung der geografischen Koordinaten des eigenen Observatoriums) leicht vollzogen werden kann.

Wie man eine vom *Betriebssystem getrennte eigene Python-Distribution* installiert, habe ich auf meiner Webseite

<https://lotharschanne.wordpress.com/>
erklärt.

Um die Skripte verwenden zu können müssen diejenigen Python-Module in der verwendeten Pythondistribution installiert sein, die am Anfang der Skripte mit den *import-Befehlen* importiert werden.

Man kann die Skripte auf verschiedenen Wegen starten:

Im einfachsten Fall geht man in eine Konsole, wechselt in das Verzeichnis, in dem die Spektrumdateien (fits oder ASCII) enthalten sind, die man bearbeiten will und startet das jeweilige Skript mit dem Befehl:

```
python3 Pfad/zu/dem/Skript/Skriptname.py
```

Eine andere Methode ist der Start eines Skripts innerhalb einer Entwicklungsumgebung. Ich bevorzuge dafür *Spyder*, das den Vorteil bietet, dass neben einer Konsole, in der die Ausgaben des Skripts erscheinen, auch ein Fenster mit dem Pythoncode des Skripts existiert (so dass man schnell z.B. Parameter anpassen oder einzelne Zeilen auskommentieren oder ändern kann). Außerdem werden im Programmablauf erzeugte Variablen im Variablenmanagerfenster angezeigt. Das kann hilfreich sein, wenn etwas schief geht und man z.B. sehen kann, bei welcher Spektrumdatei das Programm unterbrochen wurde, weil darin z.B. ein header-Eintrag fehlt. Und man sieht im Dateifenster die Dateien, welche

sich im Arbeitsverzeichnis aktuell befinden (inklusive derjenigen, die zwischenzeitlich vom laufenden Skript erzeugt wurden). Außerdem kann man in Spyder das Arbeitsverzeichnis einstellen und in die Umgebungsvariable PYTHONPATH die Pfade zu den Skripten aufnehmen, so dass die Skripte automatisch in diesen Pfadverzeichnissen gesucht werden. Es rentiert sich also, Spyder zu installieren und sich angewöhnen, damit routinemäßig alle Skripte zu laden, zu modifizieren und auszuführen.

Bei den meisten Skripten, welche Grafikfenster (z.B. Abbildungen von Spektren) erzeugen, wird am Anfang des Skripts auf ein interaktives backend (QT5Agg) umgeschaltet. D.h. die geöffneten Grafikfenster können aktiv verändert werden, wenn der Fokus auf ein Fenster gelegt wurde. Also kann das Fenster vergrößert oder verkleinert werden, oder von dem dargestellten Inhalt (das Spektrum) kann ein Ausschnitt ausgewählt werden, der Ausschnitt kann verschoben werden, die Grafik kann gespeichert werden etc. So kann man sich Details eines Spektrums genauer anschauen.

Part II.

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 und kann das Programm auch nicht ändern). Als oft verwendete Programme seien genannt IRIS³ und VSpec⁴, RSpec⁵ und BASS⁶.

Der Autor verwendet 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 die Interpretersprache *Python*.

Python hat gegenüber anderen Hochsprachen wie C++ den Vorteil einer einfacheren

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://aesesas.com/mediapool/142/1423849/data/DOCUMENTOS/BASS_Project_1_.pdf

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 III.

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 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, in Debian 12 Python 3.11. Verändert man diese Python-Bibliotheken, indem über einen typischen Befehl mit Administratorrechten wie „*sudo apt-get install 'Pythonmodul'*“ ein neuer Modul installiert 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

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/>

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 oder alternativ ein useireigenes *Python-Environment* zu verwenden. Und dann später für eigene Arbeiten immer dieses Python aufzurufen. So kommen das Betriebssystem-Python und das eigene nie in Konflikt.

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

Um ein vom Betriebssystem unabhängiges Python frei verwenden zu können gibt es drei Möglichkeiten, die in den nächsten 3 Abschnitten behandelt werden.

2.1. Anaconda

Es gibt eine Möglichkeit, in einem Schritt eine sehr umfangreiche und vom Betriebssystem unabhängige Python3-Distribution für das wissenschaftliche Arbeiten zu installieren: *Anaconda*¹². Die kostenlose Version enthält über 8000 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.

2.2. Python-Environment

Um ein *virtuelles Environment* (*venv*) in einer Linuxdistribution einzurichten benötigt das systemweite Betriebssystem das Paket *python3-venv*, das in einer Konsole mit dem Befehl

```
sudo apt-get install python3-venv
```

installiert wird.

Danach lassen sich beliebig viele *venvs* im eigenen Benutzerverzeichnis (= Homeverzeichnis) einrichten. So wird beispielsweise in einer Konsole mit dem Befehl

```
python3 -m venv /home/BENUTZER/Astro
```

im Homeverzeichnis des Benutzers namens BENUTZER ein *venv* namens Astro installiert. Hat man eine virtuelle Umgebung angelegt, muss man diese noch in einer Konsole aktivieren:

```
source /home/BENUTZER/Astro/bin/activate
```

¹²<https://www.anaconda.com/>

Nach der *Aktivierung* kann man dann mit der pythoneigenen Paketverwaltung *pip* alle weiteren benötigten Programmpakete nachinstallieren:

```
pip install spyder astropy ....
```

Um die virtuelle Umgebung zu verlassen, beendet man diese durch die Eingabe von *deactivate* am Shell-Prompt. Alternativ kann man das Terminal auch mit *exit* beenden.

Eine Deinstallation von *venvs* im eigentlichen Sinne ist nicht notwendig. Es reicht, das für die virtuelle Umgebung angelegte Verzeichnis inklusive Unterverzeichnissen zu löschen.

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.

2.3. Vollständige unabhängige Python-Version im Userhomeverzeichnis

Der folgende Vorgang ist nur für erfahrene Linuxnutzer zu empfehlen.

Falls man die Anaconda-Distribution nicht möchte oder nicht mit Environments arbeiten 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* (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
```

¹³<https://www.python.org/>

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

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

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 Programmstarttroutinen.

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

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

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

¹⁸<https://scipy.org/>

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.

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. 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. 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* oder per *pip* geladen werden können. Darunter für die Spektroskopie wichtige 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/>).

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

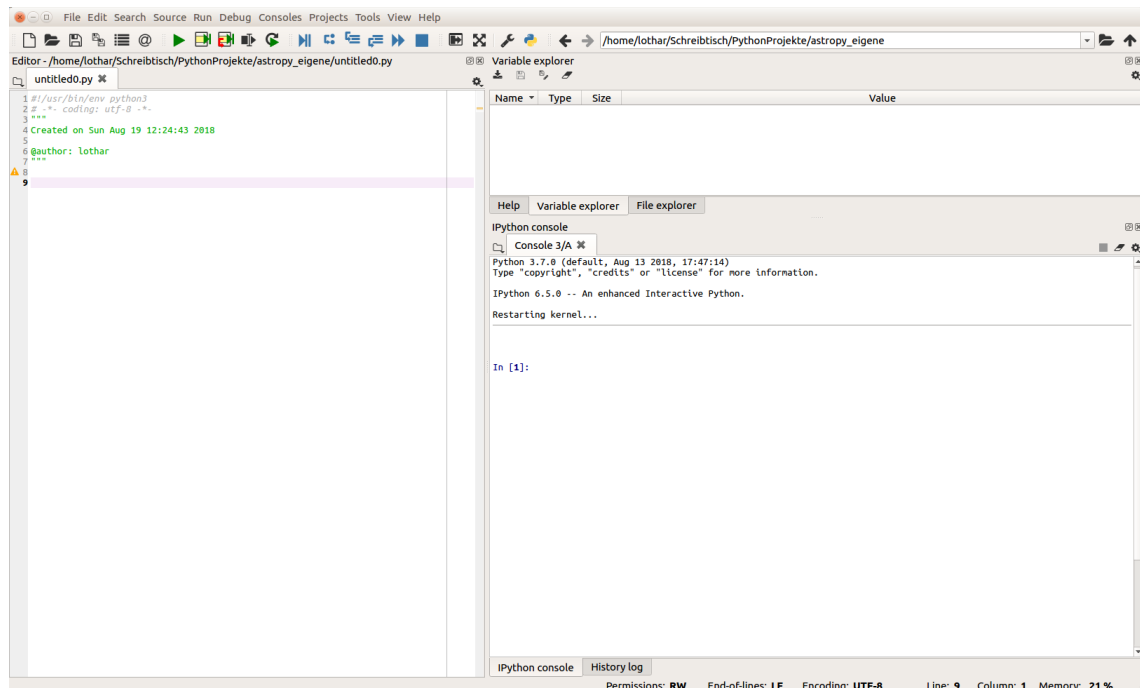


Figure 1: Spyder-Fenster nach dem Start.

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. 3 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:

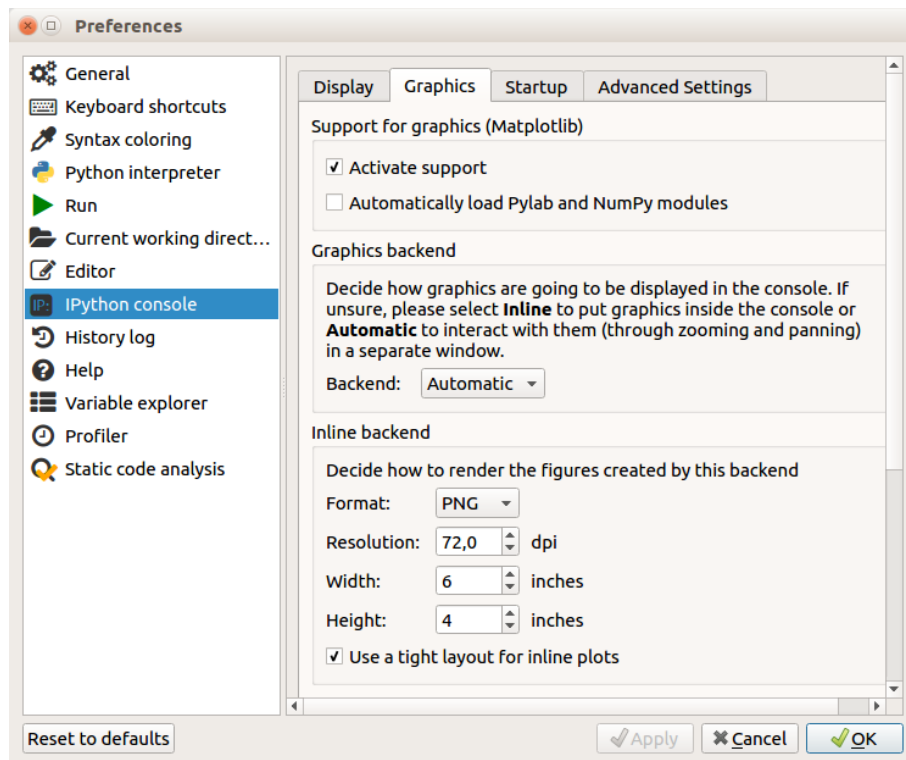
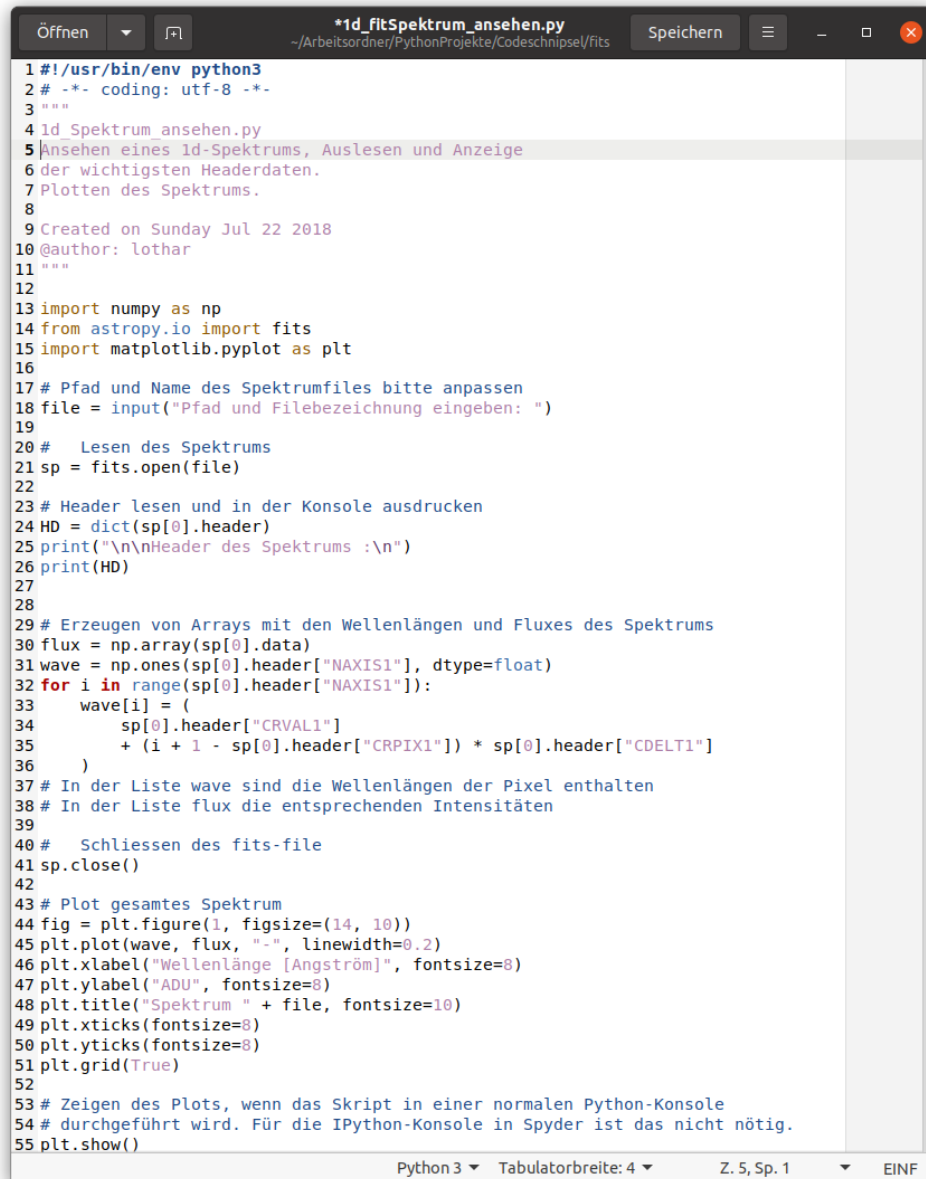


Figure 2: Einstellungen für die IPython-Konsole in Spyder.



```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
41d_Spektrum_ansehen.py
5Ansehen eines 1d-Spektrums, Auslesen und Anzeige
6der wichtigsten Headerdaten.
7Plotten des Spektrums.
8
9Created on Sunday Jul 22 2018
10@author: lothar
11"""
12
13import numpy as np
14from astropy.io import fits
15import matplotlib.pyplot as plt
16
17# Pfad und Name des Spektrumfiles bitte anpassen
18file = input("Pfad und Filebezeichnung eingeben: ")
19
20# Lesen des Spektrums
21sp = fits.open(file)
22
23# Header lesen und in der Konsole ausdrucken
24HD = dict(sp[0].header)
25print("\n\nHeader des Spektrums :\n")
26print(HD)
27
28
29# Erzeugen von Arrays mit den Wellenlängen und Fluxes des Spektrums
30flux = np.array(sp[0].data)
31wave = np.ones(sp[0].header["NAXIS1"], dtype=float)
32for i in range(sp[0].header["NAXIS1"]):
33    wave[i] = (
34        sp[0].header["CRVAL1"]
35        + (i + 1 - sp[0].header["CRPIX1"]) * sp[0].header["CDELT1"]
36    )
37# In der Liste wave sind die Wellenlängen der Pixel enthalten
38# In der Liste flux die entsprechenden Intensitäten
39
40# Schliessen des fits-file
41sp.close()
42
43# Plot gesamtes Spektrum
44fig = plt.figure(1, figsize=(14, 10))
45plt.plot(wave, flux, "-", linewidth=0.2)
46plt.xlabel("Wellenlänge [Angström]", fontsize=8)
47plt.ylabel("ADU", fontsize=8)
48plt.title("Spektrum " + file, fontsize=10)
49plt.xticks(fontsize=8)
50plt.yticks(fontsize=8)
51plt.grid(True)
52
53# Zeigen des Plots, wenn das Skript in einer normalen Python-Konsole
54# durchgeführt wird. Für die IPython-Konsole in Spyder ist das nicht nötig.
55plt.show()
```

Figure 3: Programmcode zur Ansicht eines 1d-Spektrums im fit-Format und der Header-Angaben.

- *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 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. 4. Alle Keywords des Headers sind in der Konsole rechts unten ausgegeben. Und auch alle im Programmablauf definierten Variablen werden im Variablenexplorer aufgeführt (Fenster

²⁰<http://www.numpy.org/>

²¹<http://www.astropy.org/>

²²<https://matplotlib.org/>

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*).

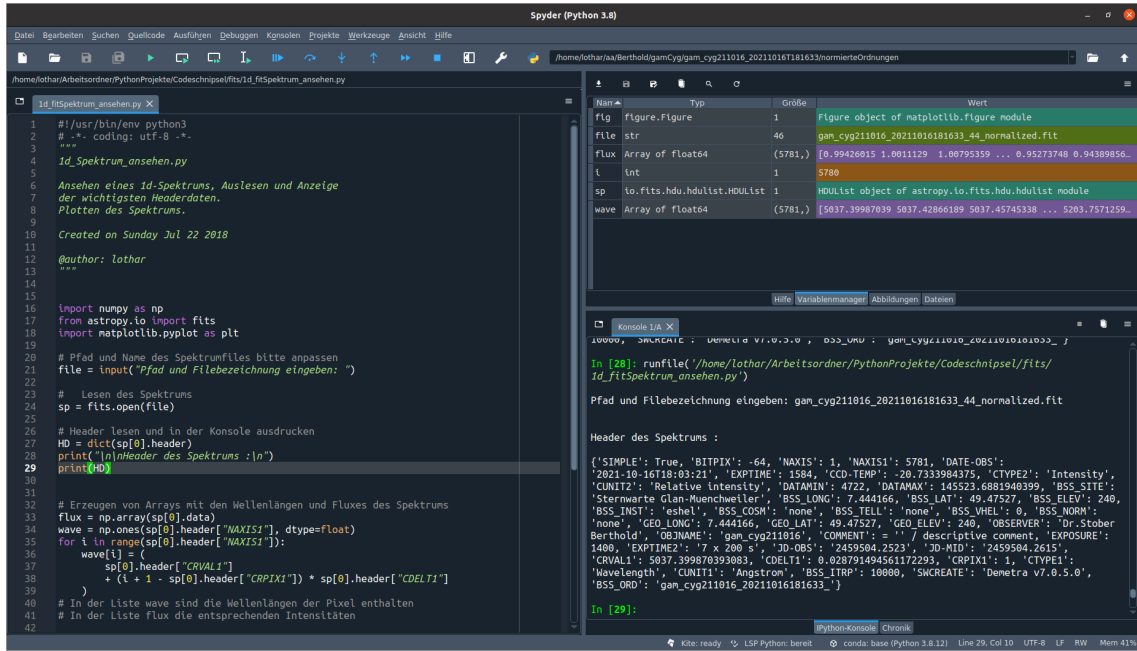


Figure 4: Ausdruck des Headers in der Konsole nach Start des Programms aus dem Editor von Spyder heraus.

rechts oben), inkl. Variablentyp, -größe und die ersten Werte dazu. Wichtig für die weiteren Berechnungen sind die Headerwerte:

1. NAXIS1 = 5781 / Axis length = Anzahl der Pixel/Wellenlängenpunkte
2. CRVAL1 = 5037.399870393083 = Start der Wellenlängenskala in Angström
3. CDEL1 = 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.

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

$$\text{Wellenlänge}[99] = \text{CRVAL1} + (99 - \text{CRPIX1} + 1) * \text{CDEL1}$$

Die Indexzählung beginnt in Python mit 0, also ist $\text{Wellenlänge}[0] = \text{CRVAL1} + (0 - \text{CRPIX1} + 1) * \text{CDEL1}$ = die Startwellenlänge des Spektrums auf der kurzwelligen Seite.

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. 5). Beachten Sie, dass wir die Einstellungen für die IPython-Konsole (siehe Abb. 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 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 IV.

Beschreibung der einzelnen Skripte

7. Ordner `ascii`

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.

7.1. `1d_dat_read_plot_csv_speichern.py`

Das Skript liest eine ASCII-Datei und plottet das Spektrum. Die ASCII-Datei hat 2 Spalten mit den Spaltenüberschriften 'WAVE' und 'FLUX'. Die Erweiterung der Datei ist `.dat`. Das Trennzeichen wird automatisch mit `guess=True` ermittelt. Plotten und Speichern als `csv` ohne Spaltenüberschriften (Skriptzeilen 40 und 41).

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

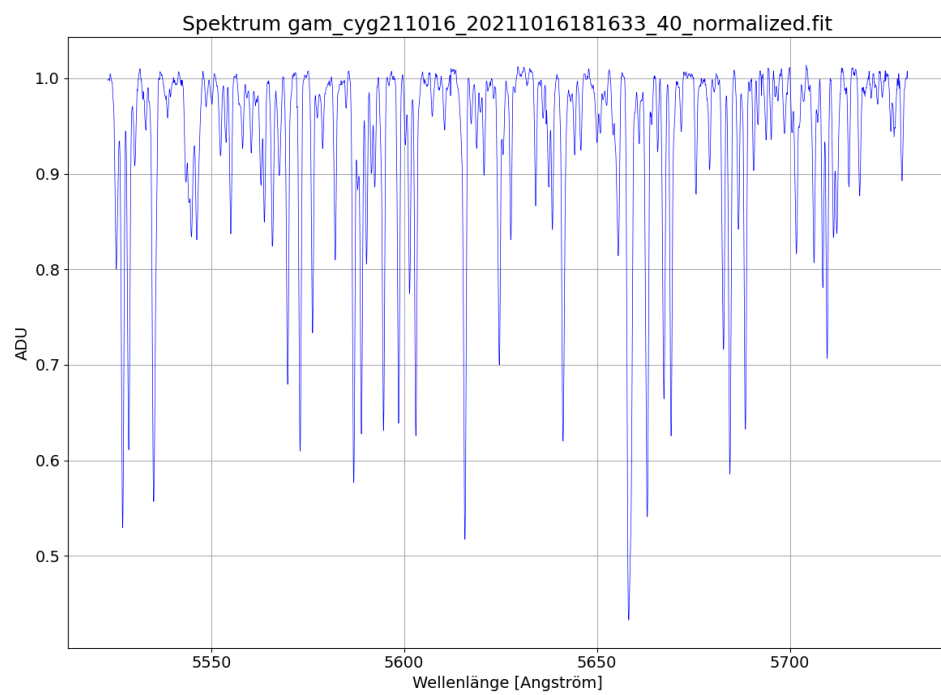


Figure 5: Durch das Skript 'spektrum_ansehen.py' erzeugter Plot eines Spektrums.

7.2. 1d_txt_plotten.py

Das Skript liest ein Spektrum, das in einer ASCII-Datei gespeichert ist. Die Daten sind in 2 Spalten mit Float-Zahlen (Wellenlänge und Fluss) ohne Überschrift. Pfad/Name werden abgefragt. Das Spektrum wird geplottet.

7.3. comparison_plot_2spectra_ascii.py

Das Skript ist gedacht um ein gemessenes Spektrum und ein Template (z.B. ein theoretisches Spektrum) übereinander zu plotten und so zu vergleichen.

Es liest das Template und das gemessene Spektrum ein (Eingabe von Pfad/Name). Die 1d-Spektren müssen als tab-separierte ASCII-Tabellen vorliegen mit den Spaltenüberschriften 'WAVE' und 'FLUX'. Der Plot wird als PDF abgespeichert (Zeile 38).

7.4. PlotAndCrop_1d_ascii.py

Das Skript hat die Aufgabe, ein im ASCII-Format vorliegendes 1d-Spektrum mit den beiden Spalten 'WAVE' und 'FLUX' zu plotten. Danach kann ein Wellenlängenbereich eingegeben werden, auf den das Spektrum reduziert wird (croppen). Dieser Ausschnitt wird dann geplottet und als ASCII-Datei gespeichert, wobei im Namen der Datei der Wellenlängenbereich genannt wird.

8. Ordner astroplanner

Die Pythonskripte dieses Ordners dienen der Planung von Beobachtungen definierter Objekte (Sterne). Sie stellen den Verlauf der Sternbahnen über dem Horizont (Sichtbarkeit) über 24 h dar.

8.1. astroplanner.py

Berechnet die Höhe des Objektes über dem Horizont für einen Zeitpunkt an einem Beobachtungsort und plottet sie über einen Zeitraum von 24 Stunden. Eingabe des Sterns, des Beobachters und des Datums nötig.

8.2. astroplanner_fuer_Objektliste.py

Berechnet die Höhen der Objekte einer Objektliste (ASCII-Datei (csv) mit den Objekt-namen in einer Spalte) über dem Horizont für einen geplanten Zeitpunkt an einem Beobachtungsort und plottet sie über einen Zeitraum von 24 Stunden. Eine Beispielobjektliste ist im Ordner enthalten. Die Grafiken werden als PDF im Arbeitsverzeichnis gespeichert. Eingabe der Liste der Sterne, des Beobachters und des Datums/Zeitpunkts nötig.

9. Ordner BC

Möchte man Radialgeschwindigkeiten aus der Doppler-Verschiebung von Spektrallinien messen, muss die jahreszeitabhängige Summe der Geschwindigkeit der Erde um die Sonne und ihre Rotationsgeschwindigkeit für den jeweiligen Beobachterort und die jeweilige Beobachtungszeit berechnet werden und die dadurch verursachte Dopplerverschiebung des gesamten Spektrums aus den Spektren heraus gerechnet werden (Heliozentrische Korrektur). Dabei wird nur die Geschwindigkeitskomponente in Richtung des Objekts berücksichtigt, weshalb auch das Objekt (dessen Koordinaten REC und DEC) für diese Korrektur bekannt sein muss.

9.1. BC_1d_spectra_fits_series.py

Das Skript berechnet die heliozentrische Korrektur BC und das korrigierte JD als HJD einer Serie von 1d-Spektren im Fits-Format eines Objekts. Schreibt eine ASCII-Datei (Format Tab) mit den BC's und HJD's der Serie in das Arbeitsverzeichnis. Korrigiert auf Wunsch die eingelesenen Spektren um die berechnete heliozentrische Korrektur BC und speichert sie als fits ab (und nach auskommentieren der Zeilen 206 bis 213 auch als ASCII-Tabelle).

Vorab *müssen* im Skript die Koordinaten des Beobachters und des Objekts eingegeben oder auskommentiert werden. Standardmäßig werden die Koordinaten des Objekts aus dem Internet eingelesen (Internetanschluss nötig), können aber auch manuell im Skript eingetragen werden (Zeilenbereich 71 bis 107).

10. Ordner binaries (Periodenbestimmung und Berechnung von Orbitalelementen)

In diesem Ordner sind Skripte gesammelt, die dazu dienen, aus den zu definierten Zeitpunkten (in JD) gemessenen und heliozentrisch korrigierten Radialgeschwindigkeiten von SB1- oder SB2- Sternsystemen die Orbitalelemente zu bestimmen. Dazu werden vom Benutzer die Zeitpunkte (JD) und die Radialgeschwindigkeiten (RV in km/s) in einer ASCII-Datei zusammengefasst, und zwar in 2 Spalten ohne Überschrift mit einem definierten einheitlichen Delimiter getrennt. Z.B. mit einem Komma (csv-Datei) oder einem tab (dat-Datei). Diese Datei mit den experimentellen Daten wird von den hier besprochenen Skripten eingelesen.

Zuerst erfolgt eine unabhängige Ermittlung der Periode mit den Skripten *PDM_Periodenanalyse.py* und/oder *Periodenanalyse.py*. Beide verwenden unterschiedliche Methoden zur Periodenbestimmung. Welche das beste Ergebnis bringt, lässt sich anhand der Streuung der Messwerte in den Phasenplots beurteilen.

Zusätzlich zur Periode P muss noch ein Zeitpunkt To festgelegt werden, der Bezugszeitpunkt, mit dem der Phasenplot berechnet wird. Der Zeitpunkt ist beliebig. Praktisch ist, den Zeitpunkt der ersten RV-Messung zu nehmen, was von den Skripten automatisch erledigt wird.

10.1. Berechnung_Phasenplot_mitBekannterPeriode

Das Skript liest eine ASCII-Datei ein mit den beiden Spalten (ohne Spalten Überschrift !) JD (Beobachtungszeitpunkte) und RV (Radialgeschwindigkeiten). Die Zeitpunkte (JD) werden mit der einzugebenden (als bekannt vorausgesetzten) Periode gefaltet und die zugehörigen RV's mit den Phasen geplottet. Der Plot wird als PDF gespeichert. Die Ergebnisse werden in einer csv-Datei abgespeichert mit den Spaltenüberschriften JD, RV und Phase.

10.2. PDM_Periodenanalyse.py

Es handelt sich um ein Werkzeug zum Auffinden von (eventuell nicht sinusförmigen) periodischen Schwankungen in Zeitreihendaten durch Minimierung der Streuung des gefalteten Datensatzes. Die Streuung wird bin-weise ermittelt.

Liest eine Datei (im csv- oder tab-Format) ein mit den beiden Spalten JD und zugehöriger Wert (z.B. RV's), aber ohne Spaltenüberschriften. Macht dann mit beiden Spalten eine Periodenanalyse (PDM-Analyse), zeigt grafisch das Periodogramm und gibt die Periode (in der Einheit Tage) aus.

Der Periodenbereich, für den die Berechnung durchgeführt werden soll, wird abgefragt.

10.3. Periodenanalyse.py

Liest eine Datei (im csv- oder tab-Format) ein mit den beiden Spalten JD und zugehöriger Wert (z.B. RV's), aber ohne Spaltenüberschriften. Macht dann mit beiden Spalten eine Periodenanalyse (Lomb-Scargle-Methode und Stringlängenmethode), zeigt grafisch die RV-Werte gegen die Zeit, das Periodogramm und den Phasenplot und gibt die Periode (in der Einheit Tage) aus.

Die Stringlängenmethode ist eine Technik zur Suche nach potenziell nicht sinusförmigen, periodischen Schwankungen in einem Datensatz. Die Idee besteht darin, die Daten mit einer Anzahl von Versuchsperioden zu falten. Aufeinander folgende Punkte in dem gefalteten Datensatz werden durch hypothetische Linien verbunden und die Gesamtlänge (die Stringlänge) wird berechnet. Die Längenstatistik nimmt ein Minimum an, wenn die Faltung einen gut geordneten Satz ergibt.

Der Periodenbereich, für den die Berechnung durchgeführt werden soll, wird abgefragt.

10.4. Phasenberechnung.py

Das Skript liest einen csv-File mit den Spalten JD und RV (ohne Spalten-Überschriften) ein. Mit den im Skript vorgegebenen Parametern Periode und T0 (anpassen !) werden dann die Beobachtungszeitpunkte gefaltet, der Phasenplot angezeigt und die Ergebnisse in einem ASCII-File (Format tab, Spalten 'JD', 'Phase', 'RV') gespeichert.

10.5. Phasenberechnung_ausSpektren_mitvorgegebenenToUndPeriode.py

Wie 4.4, nur dass die Beobachtungszeitpunkte (JD) aus einer Serie von 1d-Spektren im fits-Format eingelesen werden.

10.6. Orbitalelemente_ausJDundGemessenenRV_optimieren.py

Das Skript liest eine ASCII-Datei mit zwei Spalten ohne Überschrift mit den Zeitpunkten (JD, erste Spalte) und den RV's einer binary-Komponente (SB1 oder SB2) (RV, zweite Spalte) ein. Das Skript berechnet mit den als bekannt vorausgesetzten (unabhängig bestimmten) und abgefragten Orbitalparameter P (Periode) den Phasenplot. Für die übrigen Orbitalelemente reichen im allgemeinen die Schätzwerte der Zeilen 33 bis 36 aus. Dann werden die restlichen Orbitalparameter (der beobachteten Binärkomponente) K1, e, w1 und gamma mittels einer Kurvenfitting-Routine optimiert, damit die theoretischen RV's berechnet und im Phasenplot zusätzlich zum Vergleich mit den gemessenen RV's dargestellt. Die optimierten Orbitalparameter K1, e, w1, und gamma und ihre Standardabweichungen werden ausgedruckt.

10.7. RV_Berechnung_ausOrbitalelementenUndJD.py

Das Skript berechnet aus den als bekannt vorausgesetzten Orbitalparametern (im Skript anzupassen !) und einer Reihe von Zeitpunkten (in JD, werden aus einer ASCII-Tabelle eingelesen oder im Skript eingetragen) die theoretischen Radialgeschwindigkeiten aus. Schreibt die Ergebnisse in ein csv-File namens RVs.csv und plottet das Phasendiagramm.

11. Ordner Convolution

In diesem Ordner sind Skripte vereint, die zur Anpassung von höher aufgelösten theoretischen (berechneten) Spektren an die niedrigere Auflösung des eigenen Spektrographen benötigt werden.

11.1. convol_dat.py

Das Skript wird verwendet, um ein hoch-aufgelöstes (meist ein theoretisches) 1d-Spektrum mit einer Standardabweichung (bestimmt aus dem Apparateprofil des Spektrographen) so zu verbreitern, dass die Linienbreiten dem gemessenen Spektren entsprechen.

Einlesen eines 1d-Spektrums in Form einer ASCII-Tabelle mit der Extension .dat, 2-spaltig mit den Spaltenüberschriften 'WAVE' und 'FLUX'. Das Spektrum wird mit einer Standardabweichung convolviert (stddev in der Einheit des step = der von einem Pixel abgedeckte Wellenlängenbereich). Der convolvierte Flux wird zusammen mit der Wellenlänge in einer zweispaltigen ASCII-Tabelle abgespeichert (spacer = tab). Mit den Spaltenüberschriften 'WAVE' und 'FLUX'. Außerdem werden die beiden Spektren geplottet und die Plots als PDF und PNG gespeichert.

11.2. convol_fits.py

Einlesen eines hoch aufgelösten (theoretischen) 1d-Spektrums in Form einer fits-Datei.

Das Spektrum wird mit einer Standardabweichung, ausgedrückt durch die Auflösung R des Spektrographen, convolviert (gefaltet). Die gefaltete fits-Datei wird abgespeichert, wobei das R im Namen der Datei genannt wird.

11.3. Pollux_Ausschnitt_convolve.py

Einlesen eines synthetischen Spektrums in Form einer Tabelle (wie als .spec in der Pollux-Datenbank erhältlich, <http://pollux.graal.univ-montp2.fr/>). In dem begleitenden Pollux-File Spektrum.txt sind Startwellenlänge und step angegeben. Berechnung eines wählbaren Ausschnitts (Pandas Dataframe mit 'bereich' bezeichnet). Geplottet wird die Spalte 'NFLUX' = normierter Flux für das gesamte Spektrum und für bereich. Der Bereich wird mit einer Standardabweichung gefaltet (stddev in der Einheit des step). Der convolvierte Flux wird zusammen mit der Wellenlänge in einer zweispaltigen ASCII-Tabelle abgespeichert (spacer = tab, Spaltennamen WAVE und FLUX).

11.4. rotationsverbreiterung_fits.py

Einlesen einer Serie von 1d-Spektren im fits-Format (gedacht für synthetische Spektren zum Einrechnen der Rotationsverbreiterung und des limb-darkening-Effekts).

Nach Eingabe des limb-darkening-Koeffizienten und der Rotationsgeschwindigkeit des Sterns werden die Spektren rotationsverbreitert. Die convolvierten fits-Dateien werden abgespeichert, wobei die Dateinamen mit dem Begriff 'rotverbreitert' ergänzt werden.

12. Ordner DarstellungImGeschwindigkeitsraum

Die Skripte in diesem Ordner plotten eine Linie der eingelesenen Spektrenserie relativ zur Laborwellenlänge, wobei die Abszisse nicht durch die Wellenlänge, sondern durch die Geschwindigkeit in km/s gebildet wird. Die wählbaren Linien sind in einer Linienliste im Ordner *Bausteine* namens *Linienlisten.py* enthalten, die im Bedarfsfalle um weitere Linien erweitert werden kann.

12.1. fitsSerie_Linie_imGeschwindigkeitsraum

Einlesen einer Serie von heliozentrisch korrigierten Spektren im fits-Format, Darstellung einer wählbaren Linie im Geschwindigkeitsraum in zwei Plots: Erstens alle Spektren übereinander geplottet. Zweitens alle Spektren mit einem wählbaren offset übereinander versetzt geplottet. Die Plots können als PDF abgespeichert werden.

12.2. fitsSerie_Linie_imGeschwindigkeitsraum_JDBereich.py

Einlesen einer Serie von heliozentrisch korrigierten Spektren im fits-Format, Eingabe eines darzustellenden Zeitraums als JD Anfang und JD Ende. Darstellung einer wählbaren

Linie im Geschwindigkeitsraum in zwei Plots:

Erstens alle Spektren übereinander geplottet.

Zweitens alle Spektren mit einem wählbaren Offset übereinander versetzt geplottet.

Die Plots können als PDF und PNG abgespeichert werden.

12.3. fitsSerie_Linien_imGeschwindigkeitsraum_JDBereich.py

Einlesen einer Serie von heliozentrisch korrigierten Spektren im fits-Format, Eingabe eines Zeitraums als JD Anfang und JD Ende. Darstellung mehrerer wählbarer Linien im Geschwindigkeitsraum in einem Plot, mit einem wählbaren Offset übereinander geplottet. Die Plots können als PDF und PNG abgespeichert werden.

13. Ordner EW

In diesem Ordner sind Skripte zusammengefasst, welche der Berechnung der Äquivalentweite (EW) von Linien dienen.

13.1. Zeitserie_EW.py

Berechnet für eine Zeitserie AUF DAS KONTINUUM NORMIERTER SPEKTREN im fits-Format die Äquivalentweite einer Linie. Eingabe der Integrationsgrenzen grafisch-interaktiv oder manuell. Die EW-Berechnung setzt voraus, dass für alle Spektren der Serie das gleiche Wellenlängenintervall für die Berechnung des Integrals verwendet werden kann, also keine wesentlichen Verschiebungen des Spektrums durch RV-Änderungen stattfinden, oder wenn doch, dass die Linie isoliert ist (also der Flux = 1 im Umfeld ist).²⁶. Am besten heliozentrisch korrigierte Spektren verwenden. Die EW's werden in einer ASCII-Tabelle mit den Spalten 'Spektrum' und 'EW' abgespeichert.

13.2. Zeitserie_EW_Linientiefe_Linienwellenlaenge.py

Berechnet für eine Zeitserie AUF DAS KONTINUUM NORMIERTER SPEKTREN im fits-Format die Äquivalentweite einer Linie, sowie die Linientiefe und die Wellenlänge des Linienminimums. Eingabe der Integrationsgrenzen manuell oder grafisch.

Normierungsfehler werden durch eine Renormierungsroutine im Integrationsbereich kompensiert.

Die EW-Berechnung setzt voraus, dass für alle Spektren der Serie das gleiche Wellenlängenintervall für die Berechnung des Integrals verwendet werden kann, also keine wesentlichen RV-Änderungen stattfinden, oder wenn doch, dass die Linie isoliert ist (also der Flux = 1 im Umfeld ist). Am besten heliozentrisch korrigierte Spektren benutzen.

Das erste Spektrum wird geplottet und für 20 Sekunden angezeigt. In diesem Zeitraum ist das Grafikfenster interaktiv geschaltet, so dass man das Spektrum vergrößern kann und den Integrations-Wellenlängenbereich für die EW-Berechnung optisch aussuchen

²⁶Falls das nicht vorausgesetzt werden kann müssen die Spektren zuerst bzgl. ihrer RV's korrigiert werden.

kann. Dieses Reaktionszeitfenster von 20 Sekunden kann in Zeile 78 (`plt.pause(ZeitInSekunden)`) geändert werden.

13.3. Zeitserie_EW_grafisch_mehrereBereicheInEinerLinie.py

Berechnet für eine Zeitserie auf das Kontinuum normierter Spektren im fits-Format die Äquivalentweiten mehrerer zusammenhängender Bereiche einer Linie. Eingabe der Integrationsgrenzen grafisch-interaktiv.

Das erste Spektrum wird geplottet und für 15 Sekunden angezeigt. In diesem Zeitraum ist das Grafikfenster interaktiv geschaltet, so dass man das Spektrum vergrößern kann und den Integrations-Wellenlängenbereich für die EW-Berechnung optisch aussuchen und per Mausklicks festlegen kann. Dieses Reaktionszeitfenster von 15 Sekunden kann in Zeile 73 geändert werden.

Nach Auswahl des darzustellenden Wellenlängenbereichs für alle Spektren der Serie wird nach der Anzahl der Linienbereiche, für die die EW bestimmt werden soll, gefragt. Danach werden die einzelnen Spektren der Serie grafisch dargestellt und es wird verlangt, dass die Grenzen der getrennt auszuwertenden Bereiche per Mausklick eingegeben werden sollen. Dadurch werden die Wellenlängen der Integrationsgrenzen individuell für jedes Spektrum der Serie ermittelt. Am Ende werden die ermittelten EW's in eine ASCII-Datei geschrieben.

13.4. Zeitserie_EWs_mehrereBereicheInEinerLinie.py

Berechnet für eine auf das Kontinuum normierte Zeitserie im fits-Format die Äquivalentweite einer in Bereiche aufgeteilten Linie. Eingabe der für alle Spektren der Serie geltenden Integrationsgrenzen (in Angström) über eine Liste, die in Zeile 25 an den jeweiligen Fall anzupassen ist. Die EW-Berechnung setzt voraus, dass für alle Spektren der Serie die gleichen Wellenlängenintervalle für die Berechnung der Integrale verwendet werden können, also keine wesentlichen RV-Änderungen stattfinden. Deshalb grundsätzlich heliozentrisch korrigierte Spektren benutzen. Die ermittelten EW's werden in einer ASCII-Datei (Format csv) gespeichert.

14. Ordner fits

In diesem sind Skripte vereint welche lediglich einzelne 1d-Spektren im fits-Format plotten sollen.

14.1. 1d_fitSpektrum_ansehen.py

Auslesen und Anzeige der Headerdaten eines 1d-Spektrums im fit-Format. Plotten des Spektrums.

14.2. 1d_fitSpektrum_ansehen_mitLinienidentifikation.py

Ansehen eines wählbaren 1d-Spektrums im fits-Format, Eingabe der Linien, die im Spektrum mit einem senkrechten Strich markiert werden sollen. Die wählbaren Linien sind ab Zeile 21 in der Liste *Linien* aufgeführt. Diese Liste kann beliebig erweitert werden. Die Grafik kann auf Wunsch als PDF abgespeichert werden.

14.3. 1d_fitSpektrum_ansehen_mitLinienidentifikationPerElement.py

Ansehen eines wählbaren 1d-Spektrums im fits-Format, Eingabe der Ionen/Elemente, deren Linien im Spektrum mit einem senkrechten Strich markiert werden sollen. Die Grafik kann auf Wunsch als PDF abgespeichert werden.

14.4. 1d_fitSpektrum_ansehen_Mit_Subplots.py

Ansehen eines wellenlängenkalibrierten 1d-Spektrums, Auslesen und Anzeige der Headerdaten. Plotten des gesamten Spektrums und einer Aufteilung in eine wählbare Zahl von Ausschnitten, die dann in entsprechenden Unterplots dargestellt werden. Zusätzlich kann ein beliebiger Wellenlängenbereich ausgewählt und geplottet werden.

15. Ordner Formatumwandlungen

15.1. Formatumwandlungen.py

Umwandlung von

- Julianisches Datum in Kalenderdaten
- Kalenderdatum in Julianisches Datum
- RA und DEC in hexagesimaler Form in float
- RA und DEC von float in hexagesimale Form

16. Ordner KK (Kreuzkorrelationen)

Bei einer Kreuzkorrelation werden zwei gleiche Spektrenausschnitte (ein *Template* = z.B. theoretisches Spektrum und ein *Target* = heliozentrisch korrigiertes Objektspektrum) mit einander verglichen und gegenseitig verschoben, bis die Verschiebung mit der besten Übereinstimmung (Überdeckung) erreicht ist. Auf diese Weise kann eine Dopplerverschiebung (Radialgeschwindigkeit) gemessen werden, wobei eben nicht einzelne Linien betrachtet werden, sondern ganze Spektrumausschnitte.

Es ist meist sinnvoll, vorab mit den entsprechenden Pythonskripten des Ordners *Serien* (Crop_Ausschnitt...) relativ kleine Objekt-Spektrenausschnitte mit gleichem Wellenlängenbereich zu erzeugen, die möglichst wenige terrestrische Linien enthalten. Der Wellenlängenbereich des Template muss auf beiden Seiten mindestens 5 Å größer sein als

die Objektspektrenausschnitte (damit diese über dem Template um einen ausreichend großen Geschwindigkeitsbereich verschoben werden können).

16.1. CrossCorrelation_dat_Serie.py

Es wird eine Kreuzkorrelation einer Serie von Target-Spektren bzgl. eines Template-Spektrums durchgeführt. Alle liegen als ASCII-Datei (.dat) vor. Die Kreuzkorrelationsfunktionen werden geplottet und die Grafiken werden abgespeichert. Die berechneten RV's werden ausgedruckt und in einer ASCII-Datei gespeichert.

16.1.1. CrossCorrelation_fits_Serie.py

Es wird eine Kreuzkorrelation einer Serie von Target-Spektren bzgl. eines Template-Spektrums durchgeführt. Beide liegen als fits vor. Die Objektspektren können, müssen aber nicht heliozentrisch korrigiert sein. Die RV's und wahlweise heliozentrisch korrigierten RV's werden ausgedruckt und in einer Datei abgespeichert. Falls die heliozentrisch korrigierten RV's berechnet werden sollen, müssen die Beobachter- und die Objektkoordinaten im Skript angepasst werden. Standardmäßig werden die Sternkoordinaten aber durch Eingabe des Objektnamens (z.B. bet Ori) aus dem Internet übernommen. Dazu ist allerdings eine aktive Internetverbindung nötig.

16.2. CrossCorrelation_fits_Serie_einzelneLinien.py

Es wird eine Kreuzkorrelation einer Serie von Target-Spektren bzgl. eines Template-Spektrums durchgeführt. Beide liegen als fits vor. Es wird ein Zentralwellenlänge oder der Name einer Linie sowie ein Spektrumausschnitt für die Target-Spektren abgefragt (Umgebung einer Linie oder einer Zentralwellenlänge), der zur KK verwendet wird. Die Linienliste kann ab Zeile 139 bei Bedarf erweitert werden. Es wird die RV und wahlweise die heliozentrisch korrigierte RV berechnet. Die Daten werden in eine ASCII-Datei geschrieben. Die Stern- und Beobachterkoordinaten müssen im Skript angepasst werden, wobei die Sternkoordinaten auch aus dem Internet übernommen werden können.

17. Ordner Lichtkurven

17.1. AAVSO.py

Übernimmt die Lichtkurvendaten aus einer csv-Datei (welche z.B. von der AAVSO-Datenbank importiert wurde). Man gibt den auszuwertenden Zeitraum ein (JD Anfang und Ende) und das Programm berechnet die Tagesmittelwerte der mag-Werte. Diese werden in eine csv-Datei geschrieben und grafisch dargestellt.

18. Ordner Linienfitting

Hier sind die Skripte zusammengefasst, mit denen sich die Profile von Absorptions- oder Emissions-Linien mittels eines mathematischen Modells modellieren (fitten) lassen.

18.1. FitLinie.py

Das Skript dient der Modellierung eines Absorptions- oder Emissionslinienprofils. Verschiedene Modelle (Gauss, Voigt, SplitLorentz) können ausgewählt werden.

Das Skript lädt eine Serie auf das Kontinuum normierter Spektren im fits-Format ein.

Dann wird das erste Spektrum geplottet und man kann die zu modellierende Linie für eine in Zeile 89 definierte und änderbare Zeit entweder per Maus vergrößern und auswählen oder alternativ manuell den Wellenlängenbereich und die Wellenlänge des Linienextremums eingeben.

Bei der grafischen Auswahl der Linie muss man dreimal mit der Maus in den plot klicken, zuerst links der Linie (mit ausreichend Kontinuum), dann rechts der Linie und als drittes das ungefähre Linienminimum/maximum. Dadurch werden die betreffenden Wellenlängen ausgewählt.

Nach welchem Modell gefittet werden soll, ist wählbar. Zur Auswahl stehen Gauss, Doppelgauss, Lorentz, Voigt, Doppelvoigt. Die Ergebnisse des Fittings werden in einer Excel-Datei gespeichert.

Bei Emissionslinien sollte in den Anfangswerten für die Modelle die Amplitude positiv gewählt werden. Bei Absorptionslinien negativ.

Bei Doppelpeaks (blends zweier Linien) muss der Abstand zwischen den Peaks in den Anfangswerten der Modelle angepasst werden, damit der Erstfit (=initial fit im linken Teil der erzeugten Grafik) etwa zu dem gemessenen Linienprofil passt: in den Zeilen `center=dict(value=extremum-10)` die Zahl variieren.

Die Grafiken können auf Wunsch gespeichert werden.

18.2. FitLinie_automatischParametrisiert.py

Das Skript dient der Modellierung eines Absorptions- oder Emissionslinienprofils. Das Profil kann eines oder zwei Maxima oder Minima haben.

Verschiedene Modelle (Gauss, Voigt für symmetrische, SplitLorentz für asymmetrische Profile) können ausgewählt werden.

Das Skript lädt eine Serie auf das Kontinuum normierter Spektren im fits-Format ein. Wenn in einer Serie das Linienprofil dopplerverschoben ist oder ganz anders ist wie im ersten Spektrum kann es sein, dass das fitting nicht funktioniert. Solche Spektren müssen dann einzeln behandelt werden.

Das erste Spektrum wird geplottet und man kann die zu modellierende Linie für eine in Zeile 89 definierte und änderbare Zeit entweder per Mausklick vergrößern und auswählen oder alternativ manuell den Wellenlängenbereich und die Wellenlänge des Linienextremums eingeben.

Bei der grafischen Auswahl der Linie muss man vier mal mit der Maus in den plot klicken, zuerst links der Linie (mit möglichst viel Kontinuum), dann rechts (mit möglichst viel Kontinuum) und als drittes das ungefähre stärkere Linienminimum/maximum und als viertes das schwächere Linienminimum/maximum (oder wieder das Linienminimum/-maximum). Dadurch werden die betreffenden Wellenlängen und Flüsse ausgewählt.

Nach welchem Modell gefittet werden soll, ist wählbar. Zur Auswahl stehen Gauss,

Doppelgauss, Lorentz, Voigt, Doppelvoigt. Die Ergebnisse des Fittings werden in einer Excel-Datei gespeichert.

Die Grafiken können auf Wunsch gespeichert werden.

18.3. FitLinie_PositiverUndNegativerGauss.py

Das Skript dient der Gauss-Modellierung eines Emissionslinienprofils, das wie bei beta Lyrae aus einer gaussförmigen Emissionslinie mit einer darin enthaltenen zentralen Absorption besteht.

Das Skript lädt eine Serie auf das Kontinuum normierter Spektren im fits-Format ein. Wenn in einer Serie das Linienprofil dopplerverschoben wird oder ganz anders ist wie im ersten Spektrum kann es sein, dass das fitting nicht funktioniert. Solche Spektren müssen dann einzeln behandelt werden.

Dann wird das erste Spektrum geplottet und man kann die zu modellierende Linie für eine in Zeile 84 definierte und änderbare Zeit per Maus vergrößern und auswählen.

Bei der grafischen Auswahl der Linie muss man vier mal mit der Maus in den Plot klicken, zuerst links der Linie (mit möglichst viel Kontinuum), dann rechts (mit möglichst viel Kontinuum) und als drittes das ungefähre Linienmaximum und als viertes auf das Minimum der eingelagerten Absorption. Dadurch werden die betreffenden Wellenlängen und Flüsse ausgewählt.

Die Ergebnisse des Fittings werden in einer Excel-Datei gespeichert.

Die Grafiken können auf Wunsch gespeichert werden.

18.4. FitLinie_PositiverUndNegativerVoigt.py

Das Skript dient der Voigt-Modellierung eines Emissionslinienprofils, das wie bei beta Lyrae aus einer gaussförmigen Emissionslinie mit einer darin enthaltenen zentralen Absorption besteht.

Das Skript lädt eine Serie auf das Kontinuum normierter Spektren im fits-Format ein. Wenn in einer Serie das Linienprofil dopplerverschoben wird oder ganz anders ist wie im ersten Spektrum kann es sein, dass das fitting nicht funktioniert. Solche Spektren müssen dann einzeln behandelt werden.

Dann wird das erste Spektrum geplottet und man kann die zu modellierende Linie für eine in Zeile 84 definierte und änderbare Zeit per Maus vergrößern und auswählen.

Bei der grafischen Auswahl der Linie muss man vier mal mit der Maus in den plot klicken, zuerst links der Linie (mit möglichst viel Kontinuum), dann rechts (mit möglichst viel Kontinuum) und als drittes das ungefähre Linienmaximum und als viertes auf das Minimum der eingelagerten Absorption. Dadurch werden die betreffenden Wellenlängen und Flüsse ausgewählt.

Die Ergebnisse des Fittings werden in einer Excel-Datei gespeichert.

Die Grafiken können auf Wunsch gespeichert werden.

19. Ordner Normierung

In diesem Ordner finden sich Skripte, mit denen Serien von 1d-Spektren auf das Kontinuum normiert werden können.

19.1. `automaticNormalization_timeseries_20240115.py`

Funktioniert für ein einzelnes oder eine Serie von 1d-Spektren im Fits-Format. Beim ersten Spektrum werden die Parameter 'inter' und 'sm' optimiert. Nimmt von Pixel zu Pixel ein Intervall von 'inter' Pixel und vergleicht es mit den benachbarten Intervallen. Findet so lokale Maxima = Stützpunkte. Man kann auch mehrere Wellenlängenbereiche von der Bildung von Normalisierungspunkten ausschließen, d.h. löschen (breite Linien). Der Glättungsparameter 'sm' (Dezimalzahl) stellt die Empfindlichkeit des Splines ein der am Ende zur Normalisierung verwendet wird. $sm = 0.0$ bedeutet: Der Spline geht durch alle Gitterpunkte. Je größer sm (10.0, 100, 1000....), desto steifer ist der Spline. Optimal ist ein sm , wenn der Spline die Kurven des Kontinuums gut nachzeichnet, aber nicht in die Linien hineinragt. Danach können Sie noch Punkte löschen, die um einen bestimmten Prozentsatz größer oder kleiner als der Spline sind. Schließlich werden alle Spektren der Zeitreihen mit den optimierten Parametern normalisiert. Die normalisierten Spektren (sowie die Plots und die Normalisierungsfunktionen) werden auf Wunsch gespeichert. Außerdem werden alle gewählten Parameter in einer ASCII-Datei namens Parameterliste.txt gespeichert.

19.2. `1Punkt_Normierung.py`

Einlesen einer Serie von 1d-fits-Dateien und automatische Normierung auf den Mittelwert der Flüsse des Spektrums. Abspeichern der berechneten fits mit dem Namenszusatz `_1Pnorm`.

19.3. `Normierung_Abstandsmethode_Splinemethode_20241126.py`

Die 1d-Spektren im fits-Format werden eingelesen. Im ersten Schritt werden für die Fluxes in kleinen Intervallen eine Regression erster Ordnung (Gerade) berechnet und die Steigungen in einem Array gespeichert. Die Differenzen der Fluxes von Ende und Anfang der Intervalle werden berechnet und durch die Steigung des Intervalls geteilt (normiert) und in einem Array gespeichert. Dann werden die Elemente dieses Arrays gesucht, die ein vorgegebenes Perzentil unterschreiten und als Stützpunkte für die Normierungsfunktion gewertet werden, die anschließend mittels eines Splines berechnet wird. Dabei können bestimmte Wellenlängenintervalle (breite Balmerlinien) von der Bildung von Stützpunkten ausgeschlossen werden. Anschließend werden in einer Schleife mit mehreren Durchgängen Stützpunkte gelöscht, die zu weit oberhalb oder unterhalb des Splines liegen. Wenn keine Stützpunkte in einem Spektrum gefunden werden, bricht das Programm ab. Beispielsweise, wenn im Spektrum der Flux für viele Pixel Null oder sonst einen konstanten Wert beträgt. Für solche Spektren ist das Skript nicht geeignet.

19.4. Normierung_einzelne Linien_20241105.py

Das Skript dient der möglichst exakten und reproduzierbaren Normierung von 1d-Spektren im Umfeld einzelner Linien in Serien von 1d-Spektren im fits-Format. In Zeile 66 muss die zu normierende Linie durch ihre Wellenlänge definiert werden. Außerdem eine Liste von Wellenlängen objektspezifischer Stützpunkte (ab Zeile 70), durch die im Verlauf der Berechnungen ein Spline (oder durch Auskommentieren in Zeile 188 ein Polynom) gelegt wird, der oder das die Normierungsfunktion definiert, durch die dann die Intensitäten (flux) geteilt wird. Alternativ können die Stützpunkte auch interaktiv grafisch gewählt werden.

Die Grafiken werden gespeichert und die normierten Ordnungen als fits und ASCII-Datei abgespeichert.

Eine interaktiv angelegte Liste der Stützpunkte wird zudem als ASCII-File namens Stützpunktliste.txt zur Wiederverwendung abgespeichert. Diese kann für weitere gleichartige Normierungen per copy/paste in die Stützpunkt-Liste in Zeile 70 ff. eingetragen werden.

19.5. Normierung_findPeaks_Polynommethode_20241111.py

Die 1d-Spektren einer Serie im fits-Format werden eingelesen. Dazu werden mit einer peakfinder-Funktion lokale Peaks gefunden und als primäre Stützpunkte verwendet. Aus diesen Stützpunkten wird ein Spline berechnet, der als Normierungsfunktion dient. Dabei können bestimmte Wellenlängenintervalle (Balmerlinien) von der Bildung von Stützpunkten ausgeschlossen werden. Dann werden in einer Schleife mit mehreren Durchgängen Stützpunkte gelöscht, die zu weit oberhalb oder unterhalb des immer neu berechneten Splines liegen. Am Ende werden die verbliebenen Stützpunkte zur Berechnung einer Polynomfunktion verwendet, das als Normierungsfunktion dient.

Die normierten Spektren werden als fits und als ASCII-Datei abgespeichert.

19.6. Normierung_findPeaks_Splinemethode_20241111.py

Die 1d-Spektren einer Serie im fits-Format werden eingelesen. Dazu werden mit einer peakfinder-Funktion lokale Peaks gefunden und als primäre Stützpunkte verwendet. Aus diesen primären Stützpunkten wird ein Spline berechnet, der als Normierungsfunktion dient. Danach können bestimmte Wellenlängenintervalle (Balmerlinien) von der Bildung von Stützpunkten ausgeschlossen werden. Anschließend werden in einer Schleife mit mehreren Durchgängen Stützpunkte gelöscht, die zu weit oberhalb oder unterhalb des immer neu berechneten Splines liegen. Der letzte berechnete Spline dient als Normierungsfunktion. Die normierten Spektren werden als fits und als ASCII-Datei abgespeichert.

19.7. Normierung_Intervalle_Splinemethode_20241127.py

Die 1d-Spektren einer Serie im fits-Format werden eingelesen, in gleiche Intervalle geteilt und ein Perzentil des Flux und die mittlere Wellenlänge darin berechnet und als Stützpunkt für die Normierung gewertet. Dabei können bestimmte Wellenlängenintervalle

(breite Balmerlinien) von der Bildung von Stützpunkten ausgeschlossen werden. Aus den Stützpunkten wird ein Spline berechnet.

Die Einteilung in gleiche Intervalle stellt sicher, dass Stützpunkte im gesamten Spektrum gebildet werden, allerdings mit dem Nachteil, dass sie auch innerhalb von Linien liegen können.

Dann werden in einer Schleife mit mehreren Durchgängen Stützpunkte gelöscht, die zu weit oberhalb oder unterhalb des Splines liegen. Da mehr Punkte unterhalb des Splines gelöscht werden wie oberhalb, wandert der Spline bei jedem Schleifendurchgang in Richtung des Quasi-Kontinuums. Am Schluss werden die Intensitäten (flux) durch den Spline geteilt und damit die Normierung auf das Quasi-Kontinuum erreicht. Die Ergebnisse werden grafisch 0,1 Sekunde lang dargestellt, die Grafiken gespeichert und die normierten Ordnungen als fits und ASCII-Datei abgespeichert.

19.8. Normierung_Steigung_Krümmung_Splinemethode_20241126.py

Das Skript sollte nur auf 1d-Spektren angewendet werden, in denen alle Bereiche des Pseudokontinuums etwa die gleichen Steigungen haben.

Die 1d-Spektren einer Serie im fits-Format werden eingelesen. Das Spektrum wird in Intervalle unterteilt und die Flux-Werte des Intervalls mit einer Parabelfunktion gefittet. Wenn Steigung und Krümmung definierte Grenzen unterschreiten werden die Mittelwerte von Wellenlänge und Flux des Intervalls als primäre Stützpunkte verwendet. Aus diesen Stützpunkten wird ein Spline berechnet, der als primäre Normierungsfunktion dient. Anschließend können bestimmte Wellenlängenintervalle (Balmerlinien) von der Liste der Stützpunkte ausgeschlossen werden. Dann werden in einer Schleife mit mehreren Durchgängen Stützpunkte gelöscht, die zu weit oberhalb oder unterhalb des immer neu berechneten Splines liegen. Der letzte berechnete Spline dient als Normierungsfunktion. Die normierten Spektren werden als fits und als ASCII-Datei abgespeichert.

20. Ordner NRES

Im Ordner NRES sind Skripte enthalten, die man zur Bearbeitung von Ergebnisdateien benötigt, die von Spektren-Messungen mit den Echelle-Spektrographen der ferngesteuerten LCO-Teleskope²⁷ stammen.

Die gelieferten Datencontainer haben Namen wie *lscnrs01-fa09-20231014-0027-e92-1d.fits.fz*. Falls die gelieferte Datei die Extension .fz besitzt, muss sie zuerst entpackt

²⁷ <https://lco.global/observatory/instruments/nres/>

LCO's Network of Robotic Echelle Spectrographs (NRES) is four identical high-resolution ($R \sim 53,000$), precise (≤ 3 m/s), optical (380-860 nm) echelle spectrographs, each fiber-fed (2.58" per fiber width) simultaneously by two 1 meter telescopes and a ThAr calibration source. NRES is a single, globally-distributed observing facility, composed of four units located at our CTIO, SAAO, McDonald Observatory, and Wise Observatory sites, using up to eight 1-m telescopes.

Auf dieser Webseite ist auch der Aufbau der verwendeten Echelle-Spektrographen beschrieben.

Die LCO-Spektrographen liefern also hoch aufgelöste Spektren ($R \sim 53,000$) von Himmelsobjekten im gesamten optischen Wellenlängenbereich, die dem Nutzer in Form von Dateien in einem speziellen Format über das Internet geliefert werden.

werden. Das geschieht mit dem Programm funpack, das im Rahmen einer Installation des Pakets *cfitsio* installiert wird (siehe <https://heasarc.gsfc.nasa.gov/fitsio/>). Nach dem entpacken hat die Datei dann einen Namen wie *lscnrs01-fa09-20231014-0027-e92-1d.fits*.

Hat die Datei die Extension *.gz*, kann der Inhalt direkt mit dem Skript *NRES_auslesen.py* ausgelesen werden - ohne vorhergehendes entpacken der Datei.

Die Struktur des fits-Files kann untersucht werden mit einem fits-viewer wie beispielsweise dem Programm fv (siehe <https://heasarc.gsfc.nasa.gov/fitsio/fv/>). Danach gibt es 3 Ebenen:

1. Extension Primary: Besteht aus einem Header, der eine Unmenge von allgemeinen Informationen über den Datenfile enthält und die wir teilweise in die auszulesenden Ordnungen für spätere Verwendungen übernehmen.
2. Extension SPECTRUM: Enthält u.a. die einzelnen Ordnungen. Wichtig sind die Wellenlängen und der normierte Flux. Also die eigentlichen Spektren der Echelle-Ordnungen, diejenigen Spektren, die uns interessieren.
3. Extension CCF: Für uns unwichtige Inhalte.

20.1. NRES_auslesen.py

Das Auslesen der auf das Kontinuum normierten 1d-fits-Spektren der Echelle-Ordnungen Nr. 52 bis 119 erfolgt mit dem Pythonskript *NRES_auslesen.py* aus. Die nummerierten Ordnungen werden als fits-Dateien mit dem Namenszusatz *_normiert.fits* gespeichert. Innerhalb des Ausleseskripts werden die einzelnen Ordnungen auf eine gemeinsame Schrittweite (Wellenlängenbereich/Pixel) von 0,05 Å rebinnt. Die Spektren können zur Kontrolle mit dem Pythonskript *PlotAllerOrdnungen.py* in einem Plot grafisch dargestellt werden. Die Wellenlängenbereiche der einzelnen Ordnungen findet man in der Datei

NRES_order_wavelengths.csv

im Ordner NRES.

Nach dem erfolgreichen Auslesen der Spektren stehen die Echelle-Ordnungen Nr. 52 bis 119 als auf das Kontinuum normierte 1d-Spektren im fits-Format zur Verfügung. Sie tragen Namen wie

tlvnrs04-fa18-20220330-0016-e92-1d.fits.fz_Ordnung_110_norm.fits.

Auf der blauen Seite fängt der mit der Messung überdeckte Wellenlängenbereich mit der Ordnung 119 bei etwa 3900 Å an und endet mit der Ordnung 52 auf der roten Seite bei ca. 9050 Å.

Allerdings ist die Güte der Normierung nicht in allen Ordnungen an ihren Rändern aus physikalischen Gründen befriedigend, weshalb bei Bedarf eine Nachnormierung erfolgen kann (unter Verwendung von Skripten aus Ordner Normierung).

20.2. VacToAir.py

Die NRES-Echelle-Spektrographen messen die Spektren im Vakuum (die Optik befindet sich in einer Vakuumkammer), der Standard für Sternspektren ist aber eine Messung

in der Luft, weshalb die Wellenlängen der NRES-Spektren in diejenigen in der Luft umgerechnet werden sollten, damit sie mit Spektren aus anderen Quellen verglichen oder Linienlisten (Laborwellenlängen) aus den üblichen Quellen wie z.B. NIST²⁸ angewendet werden können. Diese Umrechnung erfolgt mit dem Pythonskript *VacToAir.py*. Die Dateinamen der umgerechneten und gespeicherten Spektren erhalten dabei den Zusatz *_air*.

20.3. PlotAllerOrdnungen.py

Möchte man sich alle Ordnungen eines Spektrums gemeinsam in einem Diagramm anschauen, kann dies mit dem Pythonskript *PlotAllerOrdnungen.py* geschehen.

20.4. heliocentricCorrection_NRES.py

Möchte man Radialgeschwindigkeiten messen, müssen die Spektren noch heliozentrisch korrigiert werden. Das kann mit dem Pythonskript *heliocentricCorrection.py* erfolgen. Damit hat man nun 1d-Spektren im fits-Format aller Ordnungen zur Verfügung, die auf das Kontinuum normiert und heliozentrisch korrigiert sind und weiter ausgewertet werden können.

21. Ordner Rebinning_gemitteltesSpektrum_ Differenzspektren

Manchmal benötigt man Spektrenserien, in denen alle 1d-Spektren den gleichen Wellenlängenbereich und die gleiche Schrittweite haben. Beispielsweise um von einem gemessenen Spektrum ein theoretisches Spektrum abziehen zu können. Das ermöglichen die Skripte dieses Ordners.

21.1. newbinning_mittleresSpektrum.py

Liest einen Spektrenkatalog ein (im fits-Format), rebinnt die Spektren und erzeugt tab-Spektren (tab-Tabelle mit den Spaltenbenennungen WAVE und FLUX in der ersten Zeile) sowie fits-Dateien, alle mit gleicher wählbarer Schrittweite und gleichem wählbarem Wellenlängenbereich. Die Art der Interpolation (linear oder per kubischem Spline) ist durch auskommentieren wählbar (Zeilen 100 bis 106). Außerdem wird ein gemitteltes Spektrum aus dem gemeinsamen Wellenlängenbereich berechnet und als tab-Tabelle abgespeichert mit den Spaltennamen WAVE und FLUX. Das gemittelte Spektrum wird auch geplottet und als fits abgespeichert.

21.2. Differenzspektren.py

Liest 1d-Spektren einer Serie ein (im ASCII-Format 'tab', gebildet mit dem Skript *newbinning_mittleresSpektrum.py*, alle verwendeten Spektren müssen den gleichen Wellen-

²⁸<https://www.nist.gov/pml/atomic-spectra-database>

längenbereich und die gleiche Schrittweite haben) und bildet Differenzspektren zum angegebenen Bezugsspektrum, die dann als tab-Datei gespeichert werden. Alle Differenzspektren werden in je einem Plot grafisch dargestellt, der auch abgespeichert wird.

21.3. Differenzspektren `_mitPlotAlle.py`

Wie 15.2. Zusätzlich wird ein Plot erstellt, in dem alle Differenzspektren übereinander geplottet sind.

22. Ordner `RV_Korrektur`

Gelegentlich benötigt man 1d-Spektren, deren Wellenlängenskala um eine bestimmte Radialgeschwindigkeit (RV) korrigiert sind. Beispielsweise um die Systemgeschwindigkeit oder eine phasenabhängige Orbitgeschwindigkeit aus den Spektren heraus zu rechnen.

22.1. `RVKorrektur_1d_spectra_fits_series.py`

Das Skript korrigiert eine Serie von 1d-Spektren im fits-Format eines Objekts um eine manuell eingegebene RV [km/s]²⁹. Schreibt die fits-Dateien und ASCII-Dateien der RV-korrigierten Spektren in das Arbeitsverzeichnis. Die jeweilige RV wird im Header des erzeugten Fits vermerkt.

22.2. `RVKorrekturPerTemplate_1d_spectra_fits_series.py`

Das Skript korrigiert eine Serie von 1d-Spektren im fits-Format eines Objekts um eine per KK im Vergleich zu einem Template berechnete RV [km/s]³⁰. Schreibt die fits-Dateien der RV-korrigierten Spektren in das Arbeitsverzeichnis. Die jeweilige RV wird im Header des erzeugten Fits vermerkt. Eine ASCII-Tabelle mit den RV's wird ebenfalls gespeichert.

22.3. `terrLinien_RVKorrektur_Regression_1d_spectra_fits_series`

Einlesen einer Spektrenserie im fits-Format. Diese dürfen nicht heliozentrisch korrigiert sein!

Das Skript korrigiert die Kalibrierung durch Bestimmung des Linienminimums von 3 bekannten terrestrischen Linien im Bereich der H α -Linie per Regression und rechnet die Spektren mit der ermittelten mittleren RV um. Abspeichern der korrigierten Spektren als fits. Zeigen der modellierten terrestrischen Linien als Grafik. Ausdrucken der ermittelten RV's. Schreiben der RV's, des Mittelwerts und der Standardabweichung in ein ASCII-File.

²⁹Vorsicht mit dem Vorzeichen der RV ! Möchte man beispielsweise das gemessene Spektrum eines Objekts mit einer Systemgeschwindigkeit von -6 km/s (das Objekt kommt auf uns zu) um diese RV korrigieren, muss man natürlich als RV manuell +6 eingeben um diese Systemgeschwindigkeit zu kompensieren.

³⁰Das Skript verschiebt das gemessene Spektrum auf die Wellenlängenskala des Templates, indem es die negative RV für die Dopplerverschiebung benutzt.

22.4. terrLinien_RVKorrektur_RBF_1d_spectra_fits_series.py

Einlesen einer Spektrenserie im fits-Format. Diese dürfen nicht heliozentrisch korrigiert sein.

Das Skript korrigiert die Kalibrierung durch Bestimmung des Linienminimums von bekannten terrestrischen Linien per RBF und Umrechnung der Spektren mit der ermittelten mittleren RV, wenn diese RV ± 3 km/s überschreitet³¹. Abspeichern der korrigierten Spektren als fits. Zeigen der modellierten terrestrischen Linien als Grafik. Ausdrucken der ermittelten RV's.

Die terrestrischen Linien müssen natürlich in den Spektren vorhanden sein. Im Skript sind 3 normalerweise wenig gestörte Wasserlinien zwischen 6543 und 6574 Angström berücksichtigt.

23. Ordner RV_Messung

Im Ordner RV_Messung sind Skripte vereinigt, deren Aufgabe es ist, an einzelnen Spektren oder an Spektrenserien Radialgeschwindigkeitsmessungen durchzuführen. Dazu dienen verschiedene Methoden:

- Die KK (Kreuzkorrelation) von Spektrumausschnitten
- Die Minimumbestimmung an einzelnen Linien mit unterschiedlichen Methoden.

Für die Minimumbestimmung an definierten Linien existiert ein Modul namens *Linienlisten.py* im Verzeichnis *Bausteine*, der gemeinsam von vielen der RV-Bestimmungsskripten verwendet wird und deshalb von den Skripten importiert werden muss. Deshalb muss das Modul *Linienlisten.py* im gleichen Verzeichnis stehen wie das jeweilige Skript oder in einem Verzeichnis, der in der Umgebungsvariablen PYTHONPATH enthalten ist, damit Python es findet. In dieses Modul können natürlich weitere Linien aufgenommen werden. Dazu muss das Modul nur entsprechend formatgerecht editiert werden.

23.1. RV_MessungAnLinie_fits_interaktiv.py

Liest einen Spektrenkatalog ein (Zeitreihe im fits-Format³²). Berechnet aus den Beobachtungszeitpunkten und den (anzupassenden !) Koordinaten des Beobachters und Objekts die heliozentrische Korrektur, zeigt die gewählte Linie in einem Plot, das Minimum der gewählten Linie ist anzuklicken, berechnet aus dem Minimum die heliozentrisch korrigierte Radialgeschwindigkeit RV_{bc}. Auf Wunsch kann auch die Systemgeschwindigkeit in der Korrektur berücksichtigt werden. Gibt die ermittelten Daten als ASCII-Dateien (comma-separiert) als csv aus.

³¹Die ± 3 km/s werden als mittlerer zu erwartender Fehler der Kalibrierung bei Amateurspektrographen angesehen. Falls dieser Fehler in Ihrem Falle anders ist, bitte in Zeile 145 korrigieren.

³²Die Spektren dürfen nicht heliozentrisch korrigiert sein, weil diese Korrektur das Skript selbst durchführt.

23.2. RV_MessungAnLinie_Zeitserie_dat_perInteraktion.py

Das Skript dient zur Ermittlung der RV einer Linie in Doppelsternsystemen (SB2), die in 2 Linien aufgespalten sein kann.

Liest den Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im csv-Format, 2 Spalten WAVE und FLUX). Auswahl der Linie und Festlegung des Linienminimums per Interaktion und Bestimmung der Radialgeschwindigkeit aus dem Minimum. Plottet alle Spektren zum markieren von bis zu 2 Linien-Minima und gibt die ermittelten Daten (RV und Apex beider Komponenten) als ASCII-Dateien (Komma-separiert als csv) aus. Die RV's sind nicht heliozentrisch korrigiert.

23.3. RV_MessungAnLinie_Zeitserie_dat_perInteraktion_regression_-2Linien.py

Wie das vorige Skript, nur dass das Minimum (die Minima der beiden Linien) rechnerisch per Regression (Grad 2, 4 oder 6) bestimmt wird.

Liest einen Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im csv-Format). Linienauswahl und Festlegung des Linienminimums per Interaktion. Bestimmt aus dem per Regression errechneten Minimum die Radialgeschwindigkeit RV. Plottet zur Kontrolle alle fittings und gibt die ermittelten Daten als ASCII-Dateien (Komma-separiert, als .csv) aus.

23.4. RV_MessungAnLinie_Zeitserie_dat_perRegression.py

Liest Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im tab-Format). Fitted die gewählte Linie per Regression nten Grades und bestimmt aus dem subpixelgenau berechneten Minimum die Radialgeschwindigkeiten RV. Plottet alle fittings und gibt die ermittelten Daten als ASCII-Dateien (Komma-separiert, als .csv) aus.

23.5. RV_MessungAnLinie_Zeitserie_fits_perGaussfit.py

Liest einen Spektrenkatalog ein (Zeitreihe von *normierten* 1d-Spektren im fits-Format, nicht heliozentrisch korrigiert!). Berechnet aus den Beobachtungszeitpunkten und den (anzupassenden) Koordinaten des Beobachters und Objekts die heliozentrischen Korrekturen, fittet die gewählte Linie per Gaussfit dreimal (gesamt und innere Bereiche) und bestimmt aus dem heliozentrisch korrigierten, subpixelgenau berechneten Minimum die heliozentrisch korrigierte Radialgeschwindigkeiten RV. Plottet alle fittings und gibt die ermittelten Daten als ASCII-Dateien (Komma-separiert, als .csv) aus.

23.6. RV_MessungAnLinie_Zeitserie_fits_perRBF.py

Liest einen Spektrenkatalog ein (Zeitreihe von normierten 1d-Spektren im fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden) Koordinaten des Beobachters und Objekts die heliozentrischen Korrekturen, fittet die gewählte Linie per Radial basis function (RBF) und bestimmt aus dem heliozentrisch korrigierten Minimum

die heliozentrisch korrigierte Radialgeschwindigkeit RV. Plottet zur Kontrolle alle fittings und gibt die ermittelten Daten als ASCII-Dateien (Komma-separiert, als .csv) aus.

23.7. RV_MessungAnLinie_Zeitserie_fits_perRegression.py

Liest einen Spektrenkatalog ein (Zeitserie von normierten 1d-Spektren im fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden) Koordinaten des Beobachters und Objekts die heliozentrische Korrektur, fitted die gewählte Linie im Minimumbereich per Regression und bestimmt aus dem (heliozentrisch korrigierten) Minimum die (heliozentrisch korrigierte) Radialgeschwindigkeiten RV und RV_bc. Plottet alle fittings und gibt ermittelte Daten als ASCII-Dateien (tab-separiert, als .dat) aus.

23.8. RV_MessungAnLinie_Zeitserie_fits_perRegression_AbsUndEmi.py

Liest einen Spektrenkatalog ein (Zeitserie von normierten 1d-Spektren im fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden) Koordinaten des Beobachters und Objekts die heliozentrische Korrektur, fitted die gewählte Linie im Minimum(Absorption)- oder Maximum(Emission)bereich per Regression und bestimmt aus dem heliozentrisch korrigierten Minimum/Maximum die heliozentrisch korrigierte Radialgeschwindigkeit RV. Plottet und speichert alle fittings und gibt die ermittelten Daten als ASCII-Datei (Komma-separiert, als .csv) aus.

23.9. RV_MessungAnLinie_Zeitserie_fits_perSpline.py

Liest einen Spektrenkatalog ein (Zeitserie von normierten 1d-Spektren im fits-Format). Berechnet aus dem Beobachtungszeitpunkt und den (anzupassenden) Koordinaten des Beobachters und Objekts die heliozentrische Korrektur, fitted die gewählte Linie per Spline und bestimmt aus dem (heliozentrisch korrigierten) Minimum die (heliozentrisch korrigierte) Radialgeschwindigkeiten RV und RV_bc. Plottet alle fittings und gibt ermittelte Daten als ASCII-Dateien (tab-separiert, als .dat) aus.

24. Ordner SNR

Die Pythonskripte in diesem Ordner sind dazu bestimmt, das SNR (Signal zu Rausch Verhältnis, signal noise ratio) von 1d-Spektren zu bestimmen. Um Spektren wissenschaftlich auswerten zu können, sollte das SNR über 100 liegen.

24.1. SNR_betasigma.py

Berechnet für eine Serie von 1d-fits-Spektren mittels des beta*sigma-Verfahrens das SNR jedes Spektrums. Falls sinnvoll die Parameter N und j im Skript anpassen (Zeilen 40 und 43). Das so bestimmte SNR ist nicht so aussagefähig wie das nächste.

24.2. SNR_fits_mehrereBereiche.py

Plottet das gewählte 1d-fits-Spektrum. Wähle die Bereiche für die SNR-Berechnung (Kontinuum) aus, in dem du mit linken Mausklicks die Grenzen setzt und mit doppeltem Drücken der Escape-Taste die jeweilige Eingabe des Bereichs beendest. Es können beliebig viele Bereiche eingegeben werden. Diese Bereiche sollten keine Linien enthalten.

24.3. verrauschen.py

Mit dem Skript kann man zusätzliches Rauschen in ein Spektrum einbringen.

25. Ordner theoretische Spektren

Häufig benötigt man zum Vergleich mit gemessenen Spektren solche, die theoretisch mit Hilfe von stellaren Atmosphärenmodellen berechnet wurden. Man findet sie im Internet, beispielsweise in der Polluxdatenbank³³ oder der Göttingen Spectral Library³⁴. Eine Übersicht über weitere Quellen findet man in

<http://svo2.cab.inta-csic.es/theory/newov2/index.php>.

Die theoretischen Spektren haben meist eine sehr hohe Auflösung R . Damit der Vergleich mit gemessenen Spektren gelingt ist deshalb in den Skripten die Möglichkeit gegeben, das theoretische Spektrum für die niedrigere Auflösung des gemessenen umzurechnen.

25.1. ascii_Spektrum_Ausschnitt_rebinned_convolve

Einlesen eines synthetischen Spektrums in Form einer ASCII-Tabelle mit 2 Spalten WAVE und FLUX überschrieben. Berechnung eines wählbaren Wellenlängenausschnitts. Dieser Bereich wird dann mit einer wählbaren Schrittweite rebinned und anschließend noch zusätzlich mit einer wählbaren FWHM (Apparateprofil) gefaltet. Geplottet wird der gewählte rebinnte Wellenlängenbereich und zusätzlich das gefaltete Spektrum. Der rebinnte Flux-Ausschnitt des ursprünglichen ASCII-Datei und der rebinnte und convolierte Flux wird zusammen mit der Wellenlänge in je einer zweispaltigen ASCII-Tabelle (spacer = Komma) und in je einer fits-Datei abgespeichert.

25.2. KunstspektrumErzeugen.py

Das Skript erzeugt ein künstliches, per gaußbroadening auf eine bestimmte FWHM, ausgedrückt durch eine Auflösung R , verbreitertes Linien-Spektrum. In den Zeilen 19 bis 24 werden die Linien, Linienstärken, Wellenlängenbereich und R definiert.

³³<https://pollux.oreme.org/>

³⁴https://phoenix.astro.physik.uni-goettingen.de/?page_id=15

25.3. Kurucz_Modell_GaussBroadened.py

Laden eines Kurucz-Sternatmosphärenmodells (1d-Spektrum) nach Auswahl von T_{eff} und $\log g$. Convolution mit einer Gaußfunktion auf Spektrographenauflösung, Auswahl des Wellenlängenbereiches und Anpassung auf eine gewünschte Schrittbreite. Speicherung als ASCII-tab-Tabelle und als fits-Datei mit dem einzugebenden Filename. Speicherung des Plot als PDF.

Um das Skript verwenden zu können muss im User-Verzeichnis ein Ordner PyAData existieren mit den Modellspektren. Der Ordner wird beim erstmaligen Aufrufen des Skripts automatisch angelegt. Am besten die Default-Werte bei den Fragen übernehmen.

25.4. Pollux_Ausschnitt_rebinned_convolve_spec.py

In der Pollux-Datenbank (<http://pollux.graal.univ-montp2.fr/>) lassen sich theoretische Spektren für die verschiedensten physikalischen Sterneigenschaften herunterladen. Wir verwenden das Format .spec.

Einlesen eines synthetischen Spektrums in Form einer Tabelle (wie als .spec in der Pollux-Datenbank erhältlich). Verwendet wird der normierte Flux. Berechnung eines wählbaren Wellenlängenausschnitts (Pandas Dataframe mit 'newtable' bezeichnet). Dieser Bereich wird dann mit einer wählbaren Schrittweite rebinned und anschließend noch zusätzlich mit einer wählbaren FWHM (Apparateprofil) gefaltet. Geplottet wird der gewählte rebinnte Wellenlängenbereich und zusätzlich das gefaltete Spektrum. Der rebinnte Flux-Ausschnitt des ursprünglichen .spec und der rebinnte und convolvierte Flux wird zusammen mit der Wellenlänge in je einer zweispaltigen ASCII-Tabelle (spacer = tab) und in je einer fits-Datei abgespeichert.

26. Ordner Serien

In diesem Ordner sind 25 Pythonskripte aufgenommen, die zur Bearbeitung von Serien von 1d-Spektren gedacht sind. Sie sind allerdings auch auf einzelne Spektren anwendbar, wenn einfach nur der komplette Namen des Spektrums genannt wird anstatt die Datei-bezeichnungen mit wildcards zu verallgemeinern.

26.1. AnimationAusPlots.py

Das Skript erzeugt eine Animation aus PNG's (z.B. Spektrenplots) im gängigen GIF-Format.

26.2. ascii_timeseries_einPlot_mitOffset

Liest eine Zeitserie von 1d-Spektren im tab-Format ein. Plottet die Spektren mit einem Offset übereinander und speichert den Plot als .png und .pdf ab

26.3. badPixFilter.py

Das Skript dient dazu einpixelige Spikes (heisse Pixel) in 1d-Spektren zu entfernen.

Fileliste erstellen für wellenlängenkalibrierte 1d_Spektren einer Serie im fits-Format. Ersatz von einzelnen Pixelwerten, die oberhalb eines *Grenzwertes* (badpixel) sind, durch den Mittelwert der Nachbarpixel und abspeichern aller korrigierten Spektren als sonst unverändertes fit. Der File-Name wird um '_badPixRemoved' ergänzt.

26.4. badPixFilter_VergleichNachbarpixel.py

Fileliste erstellen für wellenlängenkalibrierte 1d_Spektren einer Serie im fits-Format. Ersatz von einzelnen Pixelwerten, die die Nachbarpixel um einen *Faktor* überschreiten, durch den Mittelwert der Nachbarpixel und abspeichern aller korrigierten Spektren als sonst unverändertes fit. Der File-Name wird um '_badPixRemoved' ergänzt.

26.5. Crop_Ausschnitt_Zeitserie_fits.py

Fileliste erstellen für wellenlängenkalibrierte 1d_Spektren einer Zeitreihe im fits-Format. Plotten aller Spektren mit Wahl, ob die Grafik gespeichert werden soll. Ausdrucken des gemeinsam abgedeckten Wellenlängenbereichs in der Konsole. Beschneiden aller Spektren auf einen wählbaren Wellenlängenbereich und abspeichern aller beschnittenen als fits-Datei mit dem Wellenlängenbereich im Dateinamen.

26.6. Crop_Ausschnitt_Zeitserie_fits.py

Wie zuvor, nur mit einer Serie von 1d-Spektren im ASCII-Format.

26.7. DynamischerSpektrenplot

Erzeugt aus einer Folge von (heliozentrisch korrigierten) fits-1d-Spektren einen dynamischen Spektrenplot, wobei die Ordinate die Beobachtungszeitpunkte bilden (ein Headereintrag namens 'JD' muss im Header von jedem Spektrum existieren !). Die Intensitäten werden farbkodiert dargestellt. Der abgebildete Wellenlängenbereich wird gewählt.

26.8. FilesMitGeringerDispersionLoeschen_fits.py

Fileliste erstellen für wellenlängenkalibrierte 1d_Spektren einer Zeitreihe im fits-Format. Die Dispersionen (Schrittweiten, CDEL1) der Spektren werden überprüft. Wenn die Schrittweite einen in Zeile 40 festgelegten Wert überschreitet wird das betreffende Spektrum im Arbeitsverzeichnis gelöscht.

26.9. fit_Serie_in_csv_und_dat.py

Umwandeln einer Serie von wellenlängenkalibrierten 1d-Spektren im fit-Format in Textformat .csv (Komma-separiert) und .dat-Format (tab-separiert). Mit Spaltenüberschriften 'WAVE' und 'FLUX'.

26.10. HeaderanzeigeUndKorrektur_Serie.py

Erzeugung einer Spektrenliste der 1d-Spektren im fits-Format in einem Ordner, Korrektur oder Ergänzung des Headers. Es können beliebig viele Headereinträge geändert oder um neue Headereinträge ergänzt werden.

26.11. Headerprint_Serie_csv.py

Liest für eine 1d-Spektrenserie im fits-Format die Headerdaten ein und schreibt sie in getrennte ASCII-Dateien (.csv). Damit können also alle Headereinträge einer Serie von Spektren in eigenen ASCII-Dateien ausgewiesen und kontrolliert werden.

26.12. JD_EintragInHeader.py

Das Skript liest eine Serie von fits-1d-Spektren ein und berechnet aus unterschiedlichen Headereinträgen für das Beobachtungsdatum das jeweilige JD und trägt es in den Header des jeweiligen Spektrums als neuer Headereintrag 'JD' ein.

26.13. JD_InDateiname_uebernehmen.py

Das Skript schreibt das Beobachtungsdatum als JD an den Anfang der Dateinamen einer Serie von Spektren und speichert die fits mit dem neuen Namen als fits ab.

26.14. JD_und_Anfangswellenlaenge_InDateiname_uebernehmen.py

Das Skript schreibt das Beobachtungsdatum als JD und die Anfangswellenlänge an den Anfang der Dateinamen einer Serie von Spektren und speichert die fits mit dem neuen Namen als fits ab.

26.15. MittelungVon_fits.py

1d-Spektren im fits-Format. Sie müssen alle *gleiches* header['CRVAL1'], header['CDELTA1'], header['NAXIS1'], header['CRPIX1']) haben. Diese Variablen werden für jedes Spektrum ausgedruckt (Kontrolle der Gleichheit). Mit der Eingabe von 'y' werden alle Fluxes der Spektrenserie gemittelt und das mittlere Spektrum als fit abgespeichert.

26.16. timeseries_einPlot_mitOffset.py

Liest alle 1d-fits-Spektren einer Serie ein und plottet sie mit einem wählbaren Offset übereinander. Speichert den Graph als PNG und als PDF.

26.17. timeseries_einPlot_mitOffset_JDBereich.py

Liest eine Zeitserie von 1d-Spektren im fits-Format ein. Es wird ein JD-Bereich gewählt, der geplottet werden soll. Plottet die Spektren mit einem Offset übereinander und speichert den Plot als .PNG und .PDF ab.

26.18. timeseries_einPlot_mitOffset_JD_Bereich_Wellenlangenbereich

Liest eine Zeitserie von 1d-Spektren im fits-Format ein. Es ist ein Bereich des Julianischen Datums wählbar, dessen Spektren geplottet werden und ein Wellenlängenbereich. Plottet die Spektrenausschnitte mit einem Offset übereinander und speichert den Plot als .png und .pdf ab.

26.19. timeseries_einPlot_mitOffset_Wellenlangenbereich

Liest eine Zeitserie von 1d-Spektren im fits-Format ein. Es ist ein Wellenlängenbereich wählbar. Plottet die Spektrenausschnitte mit einem Offset übereinander und speichert den Plot als .png und .pdf ab.

26.20. timeseries_minUndmax.py

Liest eine Zeitserie von 1d-Spektren im fits-Format ein. Berechnet jeweils das Minimum und Maximum von Wellenlänge und Flux im Spektrum und schreibt diese Werte in eine ASCII-dat-Datei (Trennzeichen tab). Mit diesem Skript kann man also den Wellenlängenbereich und den Fluxbereich von jedem Spektrum einer Serie ermitteln.

26.21. timeseries_plot_JeEineGrafik.py

Liest alle 1d-fits-Spektren einer Serie ein und plottet sie in getrennten Grafiken. Die Plots werden als PDF gespeichert.

26.22. Ueberpruefung_JD_imHeader.py

Das Skript liest eine Serie von fits-1d-Spektren ein und überprüft, ob im Header ein Eintrag für das Beobachtungsdatum (in JD) vorhanden ist. Die Befunde werden je Spektrum ausgedruckt.

26.23. Wellenlaengenbereich_Zeitserie_fits

Fileliste erstellen für wellenlängenkalibrierte 1d_Spektren einer Zeitreihe im fits-Format. Abspeichern der filelist mit Angaben zum Beginn und Ende der Wellenlängenskala in einer ASCII-Datei (Format .tab). Ausdrucken des gemeinsamen Wellenlängenbereichs.

26.24. Zeitserie_Beobachtungszeitpunkte.py

Das Skript liest eine Serie von 1d-Spektren im fits-Format ein und gibt den Beobachtungszeitpunkt von jedem Spektrum in der Konsole aus. Außerdem wird der Wellenlängenbereich jedes Spektrums in der Konsole ausgedruckt. Am Ende wird eine ASCII-Datei (Format .tab) mit den Spektrennamen und dem Beobachtungszeitpunkt abgespeichert.

26.25. Zeitserie_Spektren_smoothen.py

Einlesen einer Serie von 1d-fits-Dateien und glätten des Fluxes mit einer bestimmten Fensterbreite. Abspeichern der geglätteten fits mit dem Namen bestehend aus aus Objekt, JD und einem gewählten Namenszusatz.