N2 - Photometrie eines offenen Sternhaufens

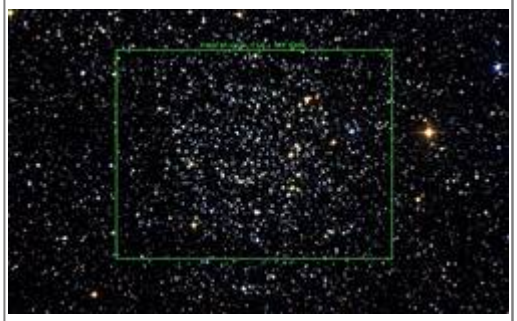
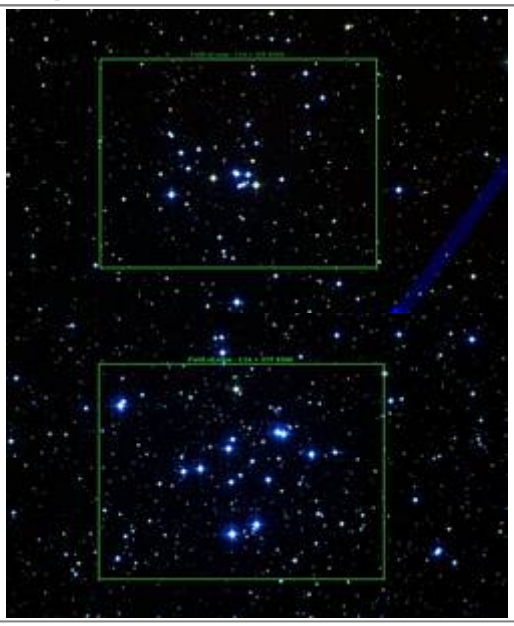
Hinweis: Dieser Artikel schildert die moderne Auswertung des N2-Versuchs mithilfe eines fast vollautomatischen *Python*-Skripts für die Reduktion von Flatfields und Darkframes sowie die Addition und Ausrichtung der Sternhaufenaufnahmen. Die klassische Version, bei der die Verschiebungen von Hand ausgemessen und manuell in eine Routine eingegeben werden, wird im Artikel Photometrie eines offenen Sternhaufens (klassisch) beschrieben. Alternativ gibt es auch noch eine halbautomatische Version auf Basis von *GDL*: Photometrie eines offenen Sternhaufens (GDL).

Aufgabe

Photometrieren sie zwei offene Sternhaufen und fertigen sie Farben-Helligkeits-Diagramme (englisch: color magnitude diagram) dieser Haufen an. Ziel der Arbeit ist die Altersbestimmung (Abschätzung) der beobachteten Sternhaufen. Zur Suche nach geeigneten Sternhaufen kann u.a. Simbad genutzt werden - eine Hilfe zur Parametersuche von Simbad gibt es hier.

Kriterien, die die Sternhaufen erfüllen müssen, sind:

- Eine hohe Zahl an Sternen im Sichtfeld, mindestens 100 Sterne sollten im Haufen vorhanden sein.
- Eine Sterndichte die nicht so hoch ist, dass die Beugungsscheibchen der Sternen miteinander verschmelzen.
- Ein großer Teil des Sternhaufens sollte im Sichtfeld der Kamera sein.
- Der Haufen sollte nicht zu klein sein.
- Es sollten keine zu starken Helligkeitsunterschiede zwischen den einzelnen Sternen im Haufen bestehen.

Gutes Beispiel	**Schlechte Beispiele**
	

Beobachtung

Der Versuch erfordert eine Nachtbeobachtung am OST der Uni Potsdam (alternativ auch am 70cm-Teleskop des AIP). Zur Vorbereitung sollte die Checkliste: Nachtbeobachtung beachtet werden.

Für die spätere Datenreduktion müssen neben den eigentlichen Aufnahmen der Sternhaufen noch Flatfields und Darks aufgenommen werden. Bias-Aufnahmen (mit null Belichtungszeit) sind nicht erforderlich, wenn zu jedem Set an Sternhaufenaufnahmen und Flats auch Darks mit den gleichen Belichtungszeiten erstellt werden. Am OST können die Flatfields nach der Beobachtung mit einer Flatfielfolie aufgenommen werden. Sollte am AIP beobachtet werden, ist es empfehlenswert die Fladsfields bereits in der Dämmerung (oder gegen eine weiße gleichmäßig beleuchtete Wand) aufzunehmen. In diesem Fall sind ebenfalls Bias-Aufnahmen erforderlich, die bei geschlossener Abdeckung zu machen sind. Auf jeden Fall sollten, bei absoluter Dunkelheit und geschlossenen Abdeckungen, die bereits erwähnten Darkframes aufgenommen werden. Zur Rauschminimierung sollte jedes Set an Darks und Flats aus mindestens 30 Einzelaufnahmen bestehen.

Nach Eintritt der Dunkelheit werden dann die Aufnahmen der Sternhaufen erstellt. Um eine ausreichende Zahl an Counts auch von leuchtschwächere Sterne zu erhalten, sollten die Aufnahmen genügend lang sein. Je nach verwendeter Kamera und beobachtetem Objekt kann diese Zeitspanne für Einzelaufnahmen zwischen 20 Sekunden und etlichen Minuten liegen. Insbesondere wenn auch hellere Objekte im Gesichtsfeld sind oder bei unzureichender Genauigkeit der Nachführung bietet es sich an mehrere Aufnahmen zu machen und diese später aufzuaddieren. Bei dem in Potsdam typischer Weise vorherschendem Seeing, können die aktuell verwendeten Kameras darüber hinaus zumeist mit einem 2×2-Binning oder 3×3-Binning betrieben werden, um so das Signal-Rausch-Verhältnisis noch weiter zu steigern. In jedem Fall sollte für diesen Versuch pro Filter eine Gesamtbelichtungszeit von mindestens 40 Minuten erreicht werden.

Datenreduktion

Vorbereitungen

Überblick verschaffen - Bilder ansehen

Als erstes steht das Einloggen im Praktikumspool an. Anschließend folgt das Kopieren der Beobachtungsdaten (FIT-Files), inklusive Flatfield- und Bias/Darkframe-Aufnahmen aus dem Verzeichnis `~/data/<datum>` ins eigene Verzeichnis `~/data_reduction/`. Es gibt verschiedene Tools, um die im FIT-Format abgelegten zweidimensionalen CCD-Bilder (Images) zu betrachten. Beispielsweise mittels *ds9*:

```
ds9 filename.fit
```

öffnet das Bild `filename.fit` mit *ds9*. Nach dem Öffnen einer Aufnahme, kann man durch Bewegen des Cursors bei gedrückter rechter Maustaste die Helligkeit und den Kontrast variieren. Weitere Optionen (Zoom, Falschfarbendarstellungen, Drehen, Spiegeln usw.) sind per Buttons zu erreichen. Die Koordinaten der aktuellen Cursor-Position werden links oben angezeigt. Man kann auch mehrere

Bilder gleichzeitig öffnen (Buttons `Frame` → `new frame`, dann weiteres File mit `File` → `open` öffnen). Mit der Blink-Option (Buttons: `Frame` → `blink`) kann man mehrere Bilder besonders gut vergleichen.

Alternativ können alle Bilder gleichzeitig geöffnet werden. Dazu wird

```
ds9 *.fit
```

eingeben (dies ist jedoch nicht für ein große Anzahl an Dateien geeignet). In *ds9* können die Frames wie oben betrachtet werden oder jeder Frame einzeln hintereinander (Button `Frame` → `Single Frame`). Dabei kann zwischen den Frames durch Drücken der Tabulatortaste umgeschaltet werden.

Es sind die **brauchbaren** Beobachtungen für die weitere Bearbeitung auszuwählen, in denen die Sterne als runde Scheibchen zu sehen sind. Bilder mit ovalförmigen Sternen sind nicht zu verwenden.

Pipeline installieren

Für die Datenreduktion und -analyse werden einige Module aus der OST-Photometrie-Pipeline benötigt. Python-Module sollten immer in einer virtuellen Umgebung installiert werden, da dies Abhängigkeitsprobleme reduziert. Eine virtuelle Umgebung kann über

```
mkvirtualenv ost_photometry
```

angelegt werden. Die virtuelle Umgebung haben wir auf diese Weise auch gleich den Namen `ost_photometry` gegeben. Das wir uns in der virtuellen Umgebung befinden lässt sich an dem String '(ost_photometry)' erkennen, der jetzt jeder Terminalzeile vorausgestellt wird. über das Kommando

```
deactivate
```

kann die virtuelle Umgebung verlassen werden. Möchte man sich wieder verbinden kann dies über

```
workon ost_photometry
```

erfolgen. Dies ist auch nötig wenn man sich, z.B. nach einer Unterbrechung der Arbeit, erneut mit columba verbindet und die Datenauswertung fortsetzen möchte.

Die OST-Photometrie-Pipeline kann anschließend in dem terminal mittels *pip* wie folgt installiert werden kann:

```
pip install ost_photometry
```

Alle benötigten Abhängigkeiten werden auf diese Weise ebenfalls installiert.

Reduktions-Pipeline: Darkframes, Flatfields und Bildstacking

Um auch mit einer größeren Menge von Daten zurecht zu kommen, gibt es eine *Python*-Routine, die pro Filter die Korrekturen für Darkframe und Flatfield durchführt, anschließend die Bilder pro Filter aufaddiert und zueinander ausrichtet. Die Routine nimmt keine Qualitätskontrolle der Bilder vor,

unbrauchbare Beobachtungen müssen also auf jeden Fall vorher aussortiert werden, sonnst kann es zu Problemen bei der Ausrichtung kommen.

Man kopiert sich zunächst das *Python*-Skript `1_add_images.py` aus dem Verzeichnis `~/scripts/n2/` in sein lokales Arbeitsverzeichnis. Danach sollte man diese mit einem beliebigen Texteditor öffnen, um die Pfadangaben für die Bilder entsprechend anzupassen. Um eine größere Menge von Bildern bequem einlesen und verifizieren zu können, erwartet das Programm eine Trennung der Daten in verschiedene Unterverzeichnisse (Variablen: `bias`, `darks`, `flats`, `images`). Es sollte jeweils ein Verzeichnis für die Aufnahmen des Sternhaufens, der Flatfields und der Darkframes existiere. Eine mögliche Verzeichnisstruktur wäre:

```
/bias/
/darks/
/flats/
/images/
```

Das *Python*-Skript erkennt automatisch die verwendeten Filter und Belichtungszeiten. Darauf aufbauend ordnet sowie klassifiziert es die Dateien automatisch ohne das weiteres Zutun nötig ist. Ist man sich sicher, dass alle `FIT-Header` Schlüsselwörter korrekt gesetzt sind können alle Datei versuchsweise auch in einem einzigen Verzeichnis abgelegt werden. In diesem Fall muss in dem Skript nur der Pfad `raw_files` gesetzt werden. Anderenfalls müssen die Pfade zu den Unterordnern bei den entsprechenden Variablen angegeben werden.

Konfigurationsbereich von `1_add_images.py`:

```
############################  Individual folders  ##############################
### Path to the bias -- If set to '?', bias exposures are not used.
bias: str = '?'

### Path to the darks
darks: str = '?'

### Path to the flats
flats: str = '?'

### Path to the images
images: str  = '?'

########################  Simple folder structure  ############################
raw_files: str = '?'
```

Wurden die Pfandangaben und der Name des Sternhaufens angepasst, kann anschließend das Skript mittels

```
 python 1_add_images.py
```

ausgeführt werden.

Die Ergebnisse liegen in einem neuen Unterverzeichnis namens `output`.

Datenauswertung mit Python

Es empfiehlt sich, alle folgende Schritte nicht auf der Kommandozeile von *Python* auszuführen, sondern ein kleines Skript auswertung.py (oder jeden anderen Namens) zu schreiben. Hierzu die gewünschten Programmdatei mittels eines Texteditors (im folgenden Fall *Kate*) öffnen:

```
kate auswertung.py &
```

Am den Beginn des *Python*-Skripts werden erst einmal die benötigten Module mit hilfreichem Code eingebunden. In unserem Fall sind das Numpy, einige Astropymodule, Astroquery sowie einige Teile unserer OST-Bibliothek:

```
import os

import numpy as np

from astropy.coordinates import SkyCoord, matching
import astropy.units as u
from astropy.table import Table

from astroquery.vizier import Vizier

from ost_photometry import checks
from ost_photometry.analyze.extraction import main_extract
from ost_photometry.analyze.plots import (
    plot_calibration_color_color_cal_stars,
    plot_combined_separation_histograms,
    plot_instrumental_vs_catalog_magnitudes,
    plot_photometry_mag_vs_error,
    plot_zeropoint_residual_distribution,
    plot_zeropoint_residual_vs_color,
    scatter,
    starmap,
)
from ost_photometry.utilities import (
    find_wcs_astrometry,
    Image,
)
from ost_photometry.analyze.utilities import (
    clear_duplicates,
)
```

```
import warnings
warnings.filterwarnings('ignore')
```

Über die letzten zwei Zeilen werden noch einige Warnungen abgeschaltet, die die Konsolenausgabe unnötig überfrachten.

Definieren einiger Variablen

Als nächstes sollten einige Variablen definiert werden. Dies sollten wenigsten der Name des Sternhaufens (`name`), das Verzeichnis in dem die Ergebnisse später abgelegt werden (`out_path`) sowie die Pfade (`V_path` und `B_path`) zu den beiden über die obige Pipeline aufaddierten und reduzierten Aufnahmen des Sternhaufens in den betrachteten Filtern (hier V und B) sein.

```
#   Cluster name (recognizable by Simbad/Vizier)
name = 'NGC7789'

#   Directory to save the data
out_path='output/'

#   Images
V_path = 'output/combined_filter_V.fit'
B_path = 'output/combined_filter_B.fit'
```

Hinweis: Die hier und im weiteren angegeben Variablennamen sind nur beispielhaft und können durch jede beliebige andere Bezeichnung ersetzt werden.

Hinweis: Liegen die Bilder nicht in einem Unterverzeichnis des aktuellen Verzeichnisses, kann mittels `../` der Pfad auch auf jeweils auf die nächsthöhere Ebene verweisen.

Einlesen der Bilder

Wir öffnen die FIT-Dateien mit Aufnahmen mittels der Funktion `image`, die über die OST-Bibliothek bereit gestellt wird. Dies hat den Vorteil, dass wir uns nicht um die Details des Einleseprozesses kümmern müssen und gleichzeitig für jedes Bild ein *Python*-Objekt zur Verfügung steht, in dem die Ergebnisse der folgenden Schritte sortiert abgelegt werden können. Die `image`-Funktion hat folgende Argumente: 1. Index des Bildes (kann hier auf `0` gesetzt werden), 2. Filterbezeichnung, 3. Pfad zur Bilddatei und 4. Pfad zum Output-Verzeichnis:

```
#   Load images
V_image = Image(0, 'V', V_path, out_path)
B_image = Image(0, 'B', B_path, out_path)
```

World Coordinate System

Die vom OST erstellten Aufnahmen werden in der Regel ohne ein sogenanntes WCS ausgeliefert. WCS steht für World Coordinate System und ermöglicht es jedem Pixel im Bild eigene Himmelskoordinaten zuzuweisen. In *ds9* werden diese dann z.B. auch beim zeigen mit dem Mauszeiger auf bestimmten Pixel bzw. auf bestimmte Objekte in dem Koordinatenfenster von *ds9* angezeigt. Dies ist sehr hilfreich wenn man z.B. die Positionen von Sternen im eigen Bild mit denen in Sternenkatalogen abgleichen will. Dies könnte bei der späteren Kalibrierung der Sternmagnituden durchaus hilfreich sein  .

Mittels der Funktion `find_wcs_astrometry` kann die Bestimmung des WCS gestartet werden:

```
#   Find the WCS solution for the images
find_wcs_astrometry(V_image)
find_wcs_astrometry(B_image)
```

Wurde eine WCS-Lösung gefunden wird auf der Kommandozeile in grün

```
  WCS solution found :)
```

ausgegeben.

Identifikation der Sterne

Finden der Sterne

Die Identifikation der Sterne in den beiden Bildern erfolgt mittels der Funktion `main_extract`. Diese Funktion nimmt als erstes Argument wiederum das `image`-Objekt. Als optionales Argument kann dann noch die Extraaktionsmethode ausgewählt werden (`photometry`). Hier spezifizieren wir `'APER'`, und wählen so Apertur-Photometrie aus, bei der der Fluss der einzelnen Objekte und der zugehörigen Himmelshintergründe innerhalb einer fest definierten (hier kreis- bzw. ringförmigen) Apertur ausgelesen wird. Um diese Apertur zu spezifizieren geben wir noch den Radius für die kreisförmige Objektapertur (`rstars`) sowie die beiden Radien für die ringförmige Hintergrundapertur (`rbg_in` und `rbg_out`) an. Bewährt haben sich hier 4 sowie 7 und 10. Die Radien sind jeweils in Bogensekunden.

```
#   Extract objects
main_extract(
    V_image,
    photometry_extraction_method='APER',
    radius_aperture=4.,
    inner_annulus_radius=7.,
    outer_annulus_radius=10.,
)
main_extract(
    B_image,
    photometry_extraction_method='APER',
    radius_aperture=4.,
    inner_annulus_radius=7.,
    outer_annulus_radius=10.
)
```

Zusätzlich zu den Sternkoordinaten (in Pixel) speichert `main_extract` auch alle extrahierten Flüsse automatisch in den `image`-Objekten.

Gefundenen Sterne prüfen

Die im vorherigen Schritt ausgeführte Funktion `main_extract` hat die schöne Eigenschaft, dass sie

die identifizierten Sterne auf einer sogenannten “Starmap” kennzeichnet. Diese kann genutzt werden, um zu prüfen, welche und im Endeffekt auch ob genügend Sterne gefunden wurden. Die Starmaps befinden sich im Ausgabeverzeichnis (Variable: `out_path`) und dort im Unterverzeichnis `starmaps`. Sollten auf der einen Seite nicht genügend Sterne identifiziert worden sein oder auf der anderen Seite neben Sternen auch Rauschen fälschlicherweise als Sterne erkannt worden sein, dann sollte im Aufruf von `main_extract` der `sigma`-Parameter angepasst werden.

Aufbereiten der Extraktionsergebnisse

Für die weiteren Schritte benötigen wir die extrahierten Flüsse und die Sternposition am Besten in Form von Astropy-Tabellen bzw. Spalten aus diesen. Die Tabellen mit den Ergebnissen erhalten wir mittels:

```
#   Get table
photo_V = V_image.photometry
photo_B = B_image.photometry
```

Aus diesen lassen sich wiederum einfach die bestimmten Sternpositionen in Pixel extrahieren:

```
x_V = photo_V['x_fit'].value.ravel()
y_V = photo_V['y_fit'].value.ravel()
```

sowie

```
x_B = photo_B['x_fit'].value.ravel()
y_B = photo_B['y_fit'].value.ravel()
```

Kreuzkorrelation und Sortierung der Ergebnisse

Nun müssen diejenigen Sterne herausgesucht werden, die in beiden Spektralbereichen/Filtern identifiziert worden sind. Dies geschieht mittels des `astropy.coordinates`-Paketes, bzw. den über diese Paket zur Verfügung gestellten Korrelations-Funktionen für Datensätze. Da diese Funktionen nicht auf Pixelkoordinaten sondern auf Basis von Himmelskoordinaten arbeiten, müssen wir unsere zuvor bestimmten Pixelkoordinaten noch in ein geeignet Koordinatensystem umrechnen. Hier ist es zum ersten Mal nützlich, dass wir zuvor das WCS bestimmt haben.

Als erstes legen wir uns `SkyCoord`-Objekte für jeden der beiden Datensätze an. Diese Objekte, wenn sie einmal definiert sind, erlauben die Ausgabe der Koordinaten in einer Vielzahl von bereits vordefinierten Koordinatensystem. Darüber hinaus, und das ist noch praktischer, werden diese Objekte auch als Argument von einer Reihe von *Astropy*-Funktionen und -Klassen akzeptiert. Aus diesem Grund braucht man sich in der Regel keine weiteren Sorgen darüber machen, in welchem Koordinatensytem man gerade eigentlich arbeitet, da dies alles intern von *Astropy* geregelt wird. Wir definieren unsere `SkyCoord`-Objekte über die Option `.from_pixel()`, welches es uns erlaubt diese direkt auf Basis der Pixelkoordinaten und des zuvor bestimmten WCS (welches wir dem `image`-Objekt entnehmen können) zu definieren.

```
#   Create SkyCoord objects
coords_V = SkyCoord.from_pixel(x_V, y_V, V_image.wcs)
coords_B = SkyCoord.from_pixel(x_B, y_B, B_image.wcs)
```

Diese beiden `SkyCoord`-Objekte können dann mittels der Funktion `search_around_sky` miteinander korreliert werden. Neben den beiden `SkyCoord`-Objekten benötigt diese Funktion als drittes Argument noch die erlaubte Toleranz in den Koordinaten (unter derer zwei Objekte aus beiden Datensatz noch als das Gleiche erkannt werden). In unserem Fall wählen wir großzügige 2 Bogensekunden. Die Einheit wird hier über das `astropy.units`-Paket definiert, dass wir oben unter der Abkürzung u geladen haben.

```
#   Correlate results from both images
id_V, id_B, d2, _ = matching.search_around_sky(coords_V, coords_B,
2.*u.arcsec)
```

Die erfolgreich zugeordneten Sterne bekommen jeweils einen Eintrag in `id_V`, `id_B` und `d2`. Diese beiden ersten Listen (genauer gesagt Numpy arrays) enthalten die Indexwerte, die diese Sterne in den ursprünglich unsortierten Datensätzen hatten. Das heißt wir können diese Indexwerte nutzen, um die ursprünglichen Tabellen mit den Flüssen und Sternpositionen so zu sortieren, dass sie nur noch Sterne enthalten die in beiden Aufnahmen detektiert wurden und das die Reihenfolge der Sterne in beiden Datensätzen die Gleiche ist. Diese Zuordnung ist essentiell für das weitere Vorgehen.

Bevor dies jedoch geschehen kann, müssen noch potentielle Mehrfachidentifikationen aussortiert werden. Es ist nämlich möglich, dass `matching.search_around_sky()` z.B. Objekt 3 aus `coords_V` sowohl Objekt 2 als auch Objekt 4 aus `coords_B` zuordnet. Das Aussortieren dieser Duplikate erfolgt mit:

```
#   Identify and remove duplicate indices
id_V, d2, id_B = clear_duplicates(
    id_V,
    d2,
    id_B,
)
id_B, _, id_V = clear_duplicates(
    id_B,
    d2,
    id_V,
)
```

Anschließend können die Tabellen mit den photometrischen Werten sortiert werden, indem die Arrays mit den Indexwerten in die entsprechenden Tabellen eingesetzt werden. Auf diese Weise selektieren und sortieren wir gleichzeitig die Sterne, die in den beiden Aufnahmen identifiziert wurden.

```
#   Sort table with extraction results and SkyCoord object
photo_V_sort = photo_V[id_V]
photo_B_sort = photo_B[id_B]

coords_objs = coords_V[id_V]
```

Neben den beiden Tabellen mit den Extraaktionsergebnissen haben wir auch eines (welches ist egal)

der `SkyCoord`-Objekte sortiert. Dies wird uns im übernächsten Schritt noch von Nutzen sein.

Umrechnung der Flüsse in Magnituden

Da im folgenden in Magnituden gearbeitet wird, müssen die Flüsse entsprechend umgerechnet werden. Die Umrechnung kann gleich auf Basis der zuvor extrahierten Tabellen erfolgen (die Flüsse sind hier in der Spalte `flux_fit` abgelegt) bzw. die Magnituden können auch gleich als neue Spalte den Tabellen hinzugefügt werden:

```
#   Calculate magnitudes
photo_V_sort['mag'] = -2.5 * np.log10(photo_V_sort['flux_fit'])
photo_B_sort['mag'] = -2.5 * np.log10(photo_B_sort['flux_fit'])
```

Man beachte, dass die Magnituden nur bis auf eine additive Konstante bestimmt sind, solange noch keine Eichung durchgeführt wurde.

Kalibrierung

Die Helligkeiten (Magnituden) sind bislang nur bis auf eine Verschiebung (additive Konstante, den sogenannten Zeropoint) bestimmt. Die Eichung (Kalibration) stellt ohne Zugang zu einer Datenbank mit Vergleichssternen ein erhebliches Problem dar. Glücklicherweise bietet die astronomische Gemeinschaft solche Datenbanken an, die wir anzapfen können. Wir werden die **VizieR**-Datenbank des **Centre de Données astronomiques de Strasbourg** benutzen bzw. unsere Kalibrationsdaten von dort beziehen. Hierfür werden wir das `astroquery`-Paket und daraus das `Vizier`-Modul benutzen.

Download Kalirationsdaten

Zuerst definieren wir den Katalog, auf den wir zugreifen wollen. In unserem Fall benutzen wir den **APASS**-Katalog, der unter der ID `II/336/apass9` läuft. Des Weiteren definieren wir die Spalten, die wir benötigen. Wir beschränken uns hier auf die Spalten, die wir wirklich brauchen, um die Downloadzeit gering zu halten. Die Spaltenbezeichnungen sind zum Teil Katalog spezifisch, sodass für einen anderen Katalog unter Umständen andere Spaltenbezeichnungen zu benutzen sind.

```
#   Get calibration from Vizier
catalog = 'II/336/apass9'
columns = ['RAJ2000', 'DEJ2000', "Bmag", "Vmag", "e_Bmag", "e_Vmag"]
```

Anschließend definieren wir das `Vizier`-Objekt. Wir übergeben diesem die Katalog-ID, die Spaltendefinition und setzen das sogenannte `row_limit` noch auf 10^6. Durch letzteres wird die herunterzuladende Tabelle auf 10^6 Zeilen und somit das Downloadvolumen begrenzt. Dies machen wir, um beim Download nicht in eine Zeitüberschreitung des Servers zu laufen.

```
v = Vizier(columns=columns, row_limit=1e6, catalog=catalog)
```

Im nächsten Schritt können wir den Download ausführen. Hierbei nutzen wir die Funktion `.query_region`. Dieser müssen wir noch die Koordinaten und die Größe des abzufragenden Himmelsbereiches übergeben. Glücklicherweise ist beides bereits bekannt. Die Koordinaten kennen wir aus den FIT-Header der Sternhaufenaufnahmen und für den Radius des abzufragenden Bereichs nehmen wir einfach das Gesichtsfeld, was wir uns bereits oben ausgerechnet haben. Beide Größen können wir z.B. dem `V_image`-Objekt entnehmen.

```
calib_tbl = v.query_region(V_image.coordinates_image_center,
radius=V_image.field_of_view_x*u.arcmin)[0]
```

Die Tabelle `calib_tbl` enthält nun alle in dem **APASS**-Katalog enthaltenen Objekte mit ihren B- und V-Magnituden, welche sich in unserem Gesichtsfeld befinden.

Aufgabe: Schränken Sie den heruntergeladenen **APASS**-Katalog auf alle Objekte im V-Magnitudenbereich von 10 bis 15 mag ein. Auf diese Weise wird sichergestellt, dass potentiell über- als auch unterbelichtete Sterne in unseren Aufnahmen nicht für die Kalibrierung verwendet werden.

Hinweis: Zur Bewältigung dieser Aufgabe kann es hilfreich sein, sich ein wenig mit booleschen Masken, Vergleichsoperation und boolescher Logik zu befassen.

Alternativ zum **APASS**-Katalog kann der “Fourth U.S. Naval Observatory CCD Astrograph Catalog” (**UCAC4**) zur Kalibrierung verwendet werden, der die ID `I/322A/` hat.

Kreuzkorrelation mit den extrahierten Daten

Anschließend muss nun der heruntergeladene Katalog mit den oben extrahierten Sternkoordinaten korreliert werden. Hierfür legen wir erneut ein `SkyCoord`-Objekt an. Diesmal für die Kalibrationssterne. Anders als oben konstruieren wir das `SkyCoord`-Objekt diesmal direkt aus den Rektaszension- und Deklinationskoordinaten, welche wir der Tabelle `calib_tbl` entnehmen können. Die Rektaszensionswerte findet man in der Spalte `RAJ2000`, wohingegen die Deklinationswerte sich in der Spalte `DEJ2000` befinden. Des Weiteren müssen noch die Einheiten für die Koordinaten angegeben werden. In unserem Fall sind das jeweils Grad (`u.deg`). Als letztes Argument (`frame`) sollte noch das Koordinatensystem definiert werden. In unserem Fall müssen wir `icrs` angeben.

```
#   Set up SkyCoord object with position of the calibration objects
coord_calib = SkyCoord(
    calib_tbl['RAJ2000'].data,
    calib_tbl['DEJ2000'].data,
    unit=(u.deg, u.deg),
    frame="icrs"
    )
```

Wie bereits oben korrelieren wir auch hier die Kalibrationsdaten mit unseren Ergebnissen mittels der`search_around_sky`-Funktion aus dem `matching`-Modul von *Astropy*. Als Argumente übergeben wir das soeben definierte `SkyCoord`-Objekt für die Kalibrationssterne, das `SkyCoord`-Objekt für die Sterne welche wir in beiden Filtern gefunden haben (`coords_objs`) sowie die maximale Distanz

zwischen Sternen in beiden Datensätzen unter derer diese noch als das selbe Objekt erkannt werden.

```
#   Correlate extracted object position with calibration table
ind_fit, ind_lit, d2, _ = matching.search_around_sky(
    coords_objs,
    coord_calib,
    2.*u.arcsec,
    )
```

Wie ebenfalls oben beschrieben, müssen nun noch die Dubletten aussortiert werden:

```
#   Identify and remove duplicate indexes
ind_fit, d2, ind_lit = clear_duplicates(
    ind_fit,
    d2,
    ind_lit,
)
ind_lit, _, ind_fit = clear_duplicates(
    ind_lit,
    d2,
    ind_fit,
)
```

Auf diese Weise erhalten wir wieder Indexwerte, die wir nutzen können, um die Kalibrationssterne sowohl aus den Datensätzen für die beiden Filter als auch aus dem heruntergeladenen Katalog zu selektieren:

```
#   Select data of the calibration stars
photo_V_sort_calib = photo_V_sort[ind_fit]
photo_B_sort_calib = photo_B_sort[ind_fit]

#   Select literature data of the calibration stars
calib_tbl_sort = calib_tbl[ind_lit]
```

Qualitätskontrolle I: Prüfen der Kalibrationssterne

Starmap

Eine Möglichkeit, die Validität der Kalibrationssterne zu prüfen ist diese sich auf einer Starmap darzustellen (ähnlich zu dem was die `main_extract` oben automatisch macht). In diesem Fall wollen wir aber die heruntergeladenen Sternpositionen als auch die Sterne darstellen, die dann später auch wirklich für die Kalibrierung verwendet wurden. Hierfür bietet die OST-Bibliothek eine geeignete Funktion (`starmap`) an, die solche Plots erstellen kann. Diese Funktion kann über

```
from ost_photometry.analyze.plots import starmap
```

eingebunden werden. Da diese Funktion als Eingabe eine Astropy-Tabellen, mit den darzustellenden Daten erwartet, müssen wir zuerst diese erstellen, bevor wir die Starmap plotten können. Die Position

der Kalibrationssterne liegen bisher nicht in Pixelkoordinaten vor, da wir diese Information von der Simbad- bzw. Vizier-Datenbank bezogen haben. Daher müssen wir zuerst diese erzeugen. An dieser Stelle ist es wieder praktisch, dass wir zuvor ein `SkyCoord`-Objekt für diese Sterne erzeugt haben. Mittels `.to_pixel()` unter Angabe des WCS des Bildes lassen sich hieraus ganz einfach Pixelkoordinaten erzeugen:

```
#   Calculate object positions in pixel coordinates
x_cali, y_cali = coord_calib.to_pixel(V_image.wcs)
```

Anschließend können wir mit diesen Information die Tabelle erstellen:

```
tbl_xy_cali_all = Table(
    names=['id','x_centroid', 'y_centroid'],
    data=[np.arange(0,len(y_cali)), x_cali, y_cali]
    )
```

Das Ganze wiederholen wir nun noch einmal für die Sterne, die wir tatsächlich als Kalibrationssterne verwenden können, also alle, die wir auch auf unseren Aufnahmen finden konnten. Hierfür erstellen wir zunächst ein entsprechendes `SkyCoord`-Objekt:

```
coord_calib_correlated = SkyCoord(
   calib_tbl_sort['RAJ2000'].data,
   calib_tbl_sort['DEJ2000'].data,
   unit=(u.deg, u.deg),
   frame="icrs",
)
```

Anschließend können wir die Pixelkoordinaten wie oben beschrieben ermitteln und eine entsprechende Tabelle anlegen:

```
x_cali_correlated, y_cali_correlated =
coord_calib_correlated.to_pixel(V_image.wcs)

tbl_xy_cali_correlated = Table(
   names=['id','x_centroid', 'y_centroid'],
   data=[np.arange(0,len(y_cali_correlated)), x_cali_correlated,
y_cali_correlated]
)
```

Daraufhin haben wir alles vorbereitet und können die Starmap plotten:

```
starmap(
   out_path,
   V_image.get_data(),
   'V',
   tbl_xy_cali_all,
   label='Downloaded calibration stars',
   tbl_2=tbl_xy_cali_correlated,
   label_2='Identified calibration stars',
   rts='calibration',
```

```
)
```

Hierbei ist das erste Argument unser Ausgabeverzeichnis, das zweite Argument das eigentlich Bild (als *Numpy*-Array), das dritte Argument die Filterbezeichnung, das vierte Argument die erste Tabelle, `label` das Label zum ersten Datensatz, `tbl_2` die zweite Tabelle, `label_2` das Label zum zweiten Datensatz und `rts` eine Beschreibung des Plots.

Alternativ kann man die Starmap auch direkt über `pyplot` aus dem `matplotlib`-Modul erstellen. Dies ist nicht viel aufwendiger bietet aber mehr Möglichkeiten zur Anpassung des Plots. Geladen wird `pyplot` mittels:

```
import matplotlib.pyplot as plt
```

Der “Grafikgrundstock” wird über

```
fig = plt.figure(figsize=(20,9))
```

erstellt. Anschließend kann das eigentliche Bild geladen werden:

```
plt.imshow(V_image, origin='lower')
```

`image` sind hierbei die eigentlichen Bilddaten und `origin=lower` stellt sicher, dass das mit dem überplotten der Koordinaten auch klappt. Daraufhin können die Symbole, die die Sternposition kennzeichnen geplottet werden:

```
plt.scatter(x_positions, y_positions)
```

`x_positions` und `y_positions` sind hier die X- bzw Y-Sternpositionen in Pixel. `.scatter` bietet eine Vielzahl an Konfigurationsmöglichkeiten wie z.B. die Auswahl des Symbols, Farbe, Linienstärke und vieles vieles mehr. Diesbezüglich verweisen wir auf die vielfältigen Dokumentation und Tutorials hierzu im Internet. Auch bezüglich Labels, Titel, Legenden und Achsenbeschriftungen lassen sich dort mehr als genug Informationen finden. Über

```
plt.savefig(filename)
```

lässt sich der Plot abspeichern. Hierbei ist `filename` der Dateiname bzw. der Pfad zur Datei. Alternativ kann der Plot über

```
   plt.show()
```

auch direkt dargestellt werden. In diesem Fall muss aber unter Umständen das Backend geändert werden bevor `plt.show()` aufgerufen wird:

```
plt.switch_backend('TkAgg')
```

Am Ende des Plots sollte dieser mittels

```
plt.close()
```

geschlossen werden.

Histogramm mit Abständen zwischen den Objekten

Mithilfe dieser Vorbereitungen können wir auch gleich ein Histogramm der Abstände zwischen den von uns gemessenen Positionen und den Positionen aus der Literatur plotten. Dies kann Hinweise auf Fehlzuordnungen zwischen den Objekten liefern.

Zunächst berechnen wir die benötigten Abstände mit Hilfe der `SkyCoord`-Objekte:

```
#   Calculate separation between objects and the calibration objects
sep_cal = coords_objs[ind_fit].separation(
   coord_calib_correlated
).arcsec

#   Calculate separation between objects on the two images (inter-filter
separations)
sep_inter = coords_V[id_V].separation(coords_B[id_B]).arcsec
```

Anschließend können wir das Histogramm plotten:

```
#   Plot separation histograms
plot_combined_separation_histograms(
   np.asarray(sep_inter, dtype=float),
   np.asarray(sep_cal, dtype=float),
   out_path,
   'pdf',
   reference_filter='V',
   other_filters=['B'],
)
```

Magnitudenkalibrierung

Nun sind wir in der Lage die eigentliche Kalibrierung der Magnituden durchzuführen. Hierfür berechnen wir den sogenannten Zeropoint, indem wir für die Kalibrationssterne in jeden der beiden Filter unsere extrahierten Magnituden von den Magnituden aus dem heruntergeladenen Katalog abziehen und anschließend mit der Funktion `.ma.median` aus dem *Numpy*-Modul über alle Kalibrationssterne den Median bilden:

```
#   Calculate zero points
ZP_V = np.ma.median(calib_tbl_sort['Vmag'] - photo_V_sort_calib['mag'])
ZP_B = np.ma.median(calib_tbl_sort['Bmag'] - photo_B_sort_calib['mag'])
```

Anschließend müssen die berechneten Zeropoints noch zu den Magnituden der Sterne in den Tabellen `photo_V_sort` und `photo_B_sort` addiert werden. Um die Übersichtlichkeit und Reproduzierbarkeit zu gewährleisten sollten die kalibrierten Magnituden jeweils in einer eigene Spalte den Tabellen hinzugefügt werden:

```
#   Calibrate magnitudes
photo_V_sort['mag_cali'] = photo_V_sort['mag'] + ZP_V
```

```
photo_B_sort['mag_cali'] = photo_B_sort['mag'] + ZP_B
```

Qualitätsskontrolle II: Stelle die instrumentellen Helligkeiten gegenüber den Kataloghelligkeiten und den Zeropoint-Abweichungen dar

Um die Qualität der Magnitudenkalibrierung einschätzen zu können, sollten wir uns die Verteilung der Magnituden sowie die Residuen der Nullpunkte visualisieren. Hierfür stellt die OST-Bibliothek einige Diagramme bereit. Um diese effizient nutzen zu können, definieren wir zunächst ein paar Abkürzungen:

```
m_inst_v = np.asarray(photo_V_sort_calib['mag'].ravel(), dtype=float)
m_inst_b = np.asarray(photo_V_sort_calib['mag'].ravel(), dtype=float)
m_cat_v = np.asarray(calib_tbl_sort['Vmag'].ravel(), dtype=float)
m_cat_b = np.asarray(calib_tbl_sort['Bmag'].ravel(), dtype=float)
m_cal_v = m_inst_v + float(zp_v)
m_cal_b = m_inst_b + float(zp_b)
```

Anschließend können wir über die in der OST-Bibliothek bereitgestellten Plots direkt die gewünschten Abbildungen zur Qualitätskontrolle erstellen:

```
#   Plot instrumental vs. catalog magnitudes (V)
plot_instrumental_vs_catalog_magnitudes(
    m_cal_v,
    m_cat_v,
    diagnostic_path,
    'pdf',
    band_label='V',
    show_one_to_one=True,
    x_label=r'$m_\mathrm{cal}$ [mag]',
    title='Cal stars: calibrated vs. catalog (V)',
    filename_stem='calibrated_vs_catalog_V',
)

#   Plot instrumental vs. catalog magnitudes (B)
plot_instrumental_vs_catalog_magnitudes(
    m_cal_b,
    m_cat_b,
    diagnostic_path,
    'pdf',
    band_label='B',
    show_one_to_one=True,
    x_label=r'$m_\mathrm{cal}$ [mag]',
    title='Cal stars: calibrated vs. catalog (B)',
    filename_stem='calibrated_vs_catalog_B',
)

#   Plot zero point residuals distribution
plot_zeropoint_residual_distribution(
    None,
```

```
    diagnostic_path,
    'pdf',
    residuals_by_band={
        'V': m_cat_v - m_inst_v - float(ZP_V),
        'B': m_cat_b - m_inst_b - float(ZP_B),
    },
    filename_stem='zeropoint_residuals_B_V',
)

#   Plot zero point residuals vs. color index
color_lit = m_cat_b - m_cat_v
plot_zeropoint_residual_vs_color(
    color_lit,
    None,
    diagnostic_path,
    'pdf',
    residuals_by_band={
        'V': m_cat_v - m_inst_v - float(ZP_V),
        'B': m_cat_b - m_inst_b - float(ZP_B),
    },
    filename_stem='zeropoint_residual_vs_color_B_V',
)

#   Plot calibration color vs. color
plot_calibration_color_color_cal_stars(
    m_cat_b - m_cat_v,
    m_inst_b - m_inst_v,
    diagnostic_path,
    'pdf',
)
```

Speichern der Ergebnisse

Ist die Kalibrierung erfolgt, sollten wir unsere extrahierten und kalibrierten Magnituden noch abspeichern. Da die Tabellen `photo_V_sort` und `photo_B_sort` einige Daten enthält, die wir nicht für die Erstellung des FHDs benötigen und wir diese der Übersichtlichkeit halber nicht mitspeichern wollen, erstellen wir uns eine neue Tabelle, die nur die relevanten Daten enthält. Die neue Tabelle kann einfach mittels `Table()` angelegt werden. Anschließend fügen wir dieser Tabelle die für uns relevanten Spalten aus den Tabellen `photo_V_sort` und `photo_B_sort` hinzu:

```
#   Create new table for the CMD
results = Table()
results['id'] = photo_V_sort['id'].ravel()
results['x'] = photo_V_sort['x_fit'].ravel()
results['y'] = photo_V_sort['y_fit'].ravel()
results['B [mag]'] = photo_B_sort['mag_cali'].ravel()
results['V [mag]'] = photo_V_sort['mag_cali'].ravel()
```

Ist die neue Tabelle fertig befüllt, erlaubt es *Astropy* diese Tabelle sehr bequem über den Befehl

`.write` zu speichern. Es muss noch als erstes Argument der Pfand bzw. Dateinamen angegeben werden, unter dem die Tabelle gespeichert werden soll. Des Weiteren spezifizieren wir noch das Format `format` (wir wählen hier `ascii`) und setzen den Parameter `overwrite` auf `True`, sodass falls wir das Skript mehrfach laufen lassen auch immer die aktuellen Daten in die Datei geschrieben werden können.

```
#   Save table
results.write(out_path + 'cmd.dat', format='ascii', overwrite=True)
```

Nachbearbeitung

Schaut man sich die Aufnahmen der Sternhaufen an wird man feststellen, das die Sternhaufen in der Regel nur einen Teil des Gesichtsfeldes einnehmen. Zumeist wird dieser Bereich zwischen 30% und 60% des Gesichtsfeldes liegen. Wir beobachten also wahrscheinlich neben den Sternhaufen eine ganze Reihe weiterer Sterne, sogenannter Feldsterne, die eigentlich nicht zu unserem Sternhaufen gehören. Auch zwischen uns und dem Sternhaufen werden sich in der Regel einige Sterne befinden. Da diese Sterne aller Wahrscheinlichkeit nach nicht zusammen mit dem zu untersuchenden Sternhaufen entstanden sind werden diese Sterne unsere Ergebnisse in Bezug auf die Altersbestimmung verfälschen bzw. diese schwieriger interpretierbar machen.

Aufgabe: Versuchen sie die Auswahl der Sterne so weit wie Möglich auf den eigentlichen Sternhaufen zu begrenzen. Sie haben hierfür zwei Möglichkeiten, die alternativ oder additiv angewendet werden können.

1. Schränken sie die Auswahl der Sterne auf z.B. 10 Bogenminuten um die zentralen Koordinaten des Sternhaufens ein
2. Laden Sie, wie oben in der Kalibrierung vorgeführt, die Daten aus dem Gaia-Archive (Katalog-ID: `I/350/gaiaedr3`) herunter und schauen Sie sich aus diesem Datensatz insbesondere die Spalten bezüglich der Eigenbewegung der Sterne an. Nutzen Sie diese Daten, um die Sternhaufenmitglieder zu selektieren.

Hinweis: Hilfreich ist es auf jeden Fall, sich Starmaps (wie unter dem Punkt “Prüfen der Kalibrationssterne” beschrieben) oder ähnliche Plots zu erstellen, die einem bei der Bewertung der Resultate helfen.

FHDs

Scheinbares FHD plotten

Zur Erstellung des FHDs steht wiederum ein *Python*-Skript zur Verfügung, in dem nur ein paar Pfade und wenige weitere Parameter angepasst werden müssen. Diese Skript bietet auch die Möglichkeit neben den Sternen auch Isochronen zu plotten. Hierauf gehen wir weiter unten genauer ein.

Zunächst sollte allerdings das entsprechende Skript `3_plot_cmd.py` aus dem Verzeichnis `~/scripts/n2/` in das lokale Arbeitsverzeichnis kopiert werden. Anschießend sollte noch der Names des Sternhaufens (`nameOfStarcluster`) gesetzt werden sowie der Pfad zu der oben gespeicherten Datei (`CMDFileName`) mit den Magnituden ergänzt werden.

Das Skriptes `3_plot_cmd.py` kann wie folgt

```
python 3_plot_cmd.py
```

ausgeführt werden. Als Ergebnis erhält man eine PDF-Datei mit dem scheinbarem FHD. Die Axenskalierung erfolgt automatisch. Da dies aufgrund von Ausreißern nicht immer ideal ist, sollte der Plotbereich über die Variablen `x_Range_apparent` und `y_Range_apparent` angepasst werden. An dieser Stelle sind die Anführungszeichen einfach durch die Axenbegrenzungen zu ersetzen, wie z.B. `x_Range_apparent = [-0.5, 2]`.

```
################################################################################
####          Configuration: modify the file in this section            ####
################################################################################

#   Name of the star cluster
nameOfStarcluster = "NGC7789"

#   Name of CMD data file
CMDFileName = "output/cmd.dat"

###
#   Plot parameter
#

#   x_Range=[xRangeMin:xRangeMax] & y_Range=[yRangeMin:yRangeMax]
#       -> x and y range to plot (change according to your data)
#       -> The plot range is automatically adjusted, if range is set to ""
#   Apparent CMD:
x_Range_apparent=["",""]
y_Range_apparent=["",""]

#   Absolute CMD:
x_Range_absolute=["",""]
y_Range_absolute=["",""]
```

Hinweis: Neben diesen Einstellungen gibt es noch eine Reihe weiterer Konfigurationsmöglichkeiten, auf die wir hier aber nicht weiter eingehen.

Rötung & absolute Magnituden

Beim Vergleich eures scheinbaren FHDs mit der Literatur wird euch allerdings auffallen, dass die Hauptreihe verschoben scheint. Dieser Effekt entsteht durch die interstellare Materie, die sich auch zwischen den Sternen unserer Milchstrasse befindet. Wie alle andere Materie auch, kann sie durch Licht angeregt werden. Diese Energie gibt sie später wieder ab, allerdings nicht auf derselben Wellenlänge, sondern niederenergetischer, also auf der roten Seite des Spektrum. Man spricht daher

bei diesem Effekt auch von Rötung, nicht zu verwechseln mit der Rotverschiebung:

$(B-V)_{0} = (B-V) - E_{(B-V)}$

$V_{0} = V - A_{V}$

Die Rötung wird mathematisch durch den Ausdruck $E_{(B-V)}$ beschrieben, den man auch als Farbexzess bezeichnet. Er beschreibt exakt den Unterschied zwischen dem hier gemessenen $(B-V)$ und dem ungeröteten, ursprünglichen Wert $(B-V)_{0}$. Die Rötung betrifft allerdings auch die Ordinate, wo der Korrekturterm mit A_{V} bezeichnet wird. Allerdings sind A_{V} und $E_{(B-V)}$ nicht unabhängig voneinander, sondern lassen sich mit R_V ineinander umrechnen:

$A_{V} = R_V \cdot E_{(B-V)}$

In der Sonnenumgebung wird R_V oft auf 3.1 gesetzt (Seaton 1979). Damit muss also nur noch der Wert für $E_{(B-V)}$ bekannt sein, um die nötigen Korrekturen komplett durchzuführen. Entsprechende Werte findet ihr via *Simbad* über die mit einem Objekt assoziierten Paper (Veröffentlichungen) oder direkt über Datenbanksuche bei *VizieR*. Vergesst in jedem Fall nicht, den benutzten Wert und die genaue Quelle (d.h. in der Regel das direkt benutzte oder zu einem *VizieR*-Eintrag gehörende Paper) in eurem Protokoll anzugeben.

Abschließend sollten noch die scheinbaren Magnituden in absolute Magnituden konvertiert werden, damit später ein Vergleich mit Isochronen möglich ist. Hierfür muss das entsprechend Distanzmodul oder die Entfernung des Sternhaufens aus Papern (Veröffentlichungen) herausgesucht werden und die entsprechende Korrektur vorgenommen werden.

Absolutes FHD plotten

Nachdem das $E_{(B-V)}$ und die Entfernung bzw. das Distanzmodul für den entsprechenden Sternhaufen herausgesucht wurden, können diese bei den entsprechenden Variablen im Skript `3_plot_cmd.py` eingetragen werden. `m_M` ist hierbei das Distanzmodul. Die übrigen Variablen sollten selbsterklärend sein. Ist entweder `m_M` oder `distance` gegeben, wird von dem Skript neben dem scheinbaren FHD auch das absolute FHD erstellt. Sollte es nötig sein R_V anzupassen kann auch dies vorgenommen werden.

```
#   EB-V of the cluster
eB_V = 0.

#   R_V
RV = 3.1

#   Give either distance modulus of the cluster or the distance in kpc
m_M = '?'

distance = '?'
```

Hinweis: Neben diesen Einstellungen gibt es noch eine Reihe weiterer Konfigurationsmöglichkeiten, auf die wir hier aber nicht weiter eingehen.

Isochronen plotten

Einige Isochronen sind bereits in der OST-Bibliothek enthalten, obwohl längst nicht alle und teilweise sind diese auch nicht vollständig. Von daher sollte, insbesondere wenn keine passenden Isochronen gefunden wurden, selbstständig nach weiteren gesucht werden. Sternentwicklungsrechnungen werden von einer Reihe von Arbeitsgruppen bzw. Wissenschaftlern durchführt. Die daraus resultieren Isochronen werden der wissenschaftlichen Gemeinschaft zumeist über Webportale zur Verfügung gestellt und können von dort herunter geladen werden.

Bedauerlicherweise gibt es kein einheitliches Format für Isochronen, was zur Folge hat, dass dem Skript (`3_plot_cmd.py`) für jeden neuen “Isochronentyp” bzw. jede neue “Isochronenquelle” beigebracht werden muss diese zu lesen. Dies erfolgt über Dateien im sogenannte *YAML*-Format, in denen die nötige Konfiguration abgelegt ist. Für die in der OST-Bibliothek enthalten Isochronen sind diese Konfigurationsdateien bereits in dem Skriptverzeichnis zu finden. Ein lehre Template-Datei ist dort ebenfalls vorhanden.

Im Skript erfolgt die Auswahl der jeweiligen “Isochronenquelle” über die Variable `isochrone_configuration_file`. Hier ist der Name bzw. der Pfad zu der jeweiligen *YAML*-Datei einzutragen.

Keine Isochronen darstellen

Sollen keine Isochronen dargestellt werden muss `isochrone_configuration_file` auf

```
 ""
```

gesetzt werden.

Für die PARCEC-Isochronen sieht die Konfigurationsdatei z.B. so aus:

```
---
#   PARCES isochrones (CMD 3.6)

#   Files
# isochrones:
'~/isochrone_database/parsec_iso/3p6/solar_0p2Gyr/iso_parsec_0p2Gyr.dat'
# isochrones:
'~/isochrone_database/parsec_iso/3p6/solar_0p5Gyr/iso_parsec_0p5Gyr.dat'
isochrones:
'~/isochrone_database/parsec_iso/3p6/solar_1Gyr/iso_parsec_1Gyr.dat'

#   Type
isochrone_type: 'file'

#   Type of the filter used in CMD plots
#   Format:
#     'filter name':
#       - column type (single or color)
#       - ID of the filter if the column type is color, e.g., if the filter
```

```
is
#         R and the color is V-R, the filter ID would be 1. If column-type
is
#         single, the ID will be 0.
#       - name of the second filter, in the example above it would be V. If
#         column-type is single, the name can be set to '-'.
isochrone_column_type:
    'U':
      - 'single'
      - 0
      - '-'
    'B':
      - 'single'
      - 0
      - '-'
    'V':
      - 'single'
      - 0
      - '-'
    'R':
      - 'single'
      - 0
      - '-'

#   ID of the columns in the isochrone data file containing the magnitudes
#   and the age
isochrone_column:
    'U': 29
    'B': 30
    'V': 31
    'R': 32
    'AGE': 3

#   Keyword to identify a new isochrone
isochrone_keyword: '# Zini'

#   Logarithmic age
isochrone_log_age: true

#   Plot legend for isochrones?
isochrone_legend: true
...
```

`isochrones` verweist auf die Datei mit den Isochronen. Hier ist `isochrone_type` auf `file` gesetzt, was dem Skript sagt, dass alle Isochronen in einer Datei zu finden sind. Eine Alternative ist `directory`. In diesem Fall erwartet das Skript, dass die Isochronen in einzelnen Dateien in einem bestimmten Verzeichnis zu finden sind und dass die Variable `isochrones` auf dieses Verzeichnis zeigt. Mit `isochrone_column` kann man die gewünschten Spaltennummern angeben. `isochrone_column_type` gibt an, ob die Größen als Farben oder als “einzelne” Magnituden

angegeben werden. Weitere Informationen findet man in der obigen Formatbeschreibung. Die grundlegenden Optionen sind hier `color` und `single`. Mit `isochrone_log_age` kann man angeben, ob die Werte in der Altersspalte logarithmiert sind oder nicht. Man kann zwischen `True` und `False` wählen. Wenn sich die Isochronen alle in einer Datei befinden, benötigt das Skript ein Schlüsselwort, um zu erkennen, wann eine Isochrone endet und die nächste beginnt. Dies kann mit der Variablen `isochrone_keyword` angegeben werden. Schließlich kann man entscheiden, ob eine Legende für die Isochronen gezeichnet werden soll. Dies wird durch die Variable `isochrone_legend` gesteuert.

Tip: Zumeist gibt es Isochronen aus einer Quelle in unterschiedlichen zeitlichen Auflösungen und für unterschiedliche Metallizitäten. Diese finden sich dann in der Regel in anderen Dateien bzw. Ordnern, je nachdem was für `isochrone_type` gesetzt werden muss. Es kann sich also lohnen, in der Datenbank nachzuschauen und den Eintrag für `isochrones` anzupassen.

Hinweis: Ein paar zusätzliche Informationen zu den einzelnen Variablen findet sich noch im *YAML*-Template.

Protokoll

Es ist ein übliches Protokoll einzureichen. Eine allgemeine Übersicht über den erforderlichen Aufbau und Inhalt findet man hier.

Beschreiben Sie im Methodenteil die Beobachtungen und die Datenreduktion, heben Sie insbesondere Punkte hervor, die von der allgemeinen Beschreibung hier abweichen, und führen Sie alle Parameter auf, die Sie für die Extraktion gesetzt haben. Fügen Sie außerdem **alle Diagramme und Grafiken** der Datenreduktion in den Bericht ein. Geben Sie auch alle Parameter für Rötung, Extinktion und Entfernung an, die Sie aus der Literatur übernommen haben.

Im Ergebnisteil werden die CMDs des Sternhaufens dargestellt und die darin beobachtbaren Merkmale beschrieben.

Die Analyse der CMDs enthält die Schätzung des Haufenalters auf der Grundlage des Abbiegepunkts und einer Isochronenanpassung.

Diskutieren Sie schließlich Ihre Ergebnisse. Stellen Sie Ihre Ergebnisse in einen größeren Zusammenhang und nehmen Sie, wenn möglich, einen Literaturvergleich vor (z. B. für das Alter des Haufens). Dazu gehört auch, dass Sie mögliche Probleme mit den Daten, der Datenreduktion oder der Analyse (insbesondere der Isochronenanpassung) und mögliche Lösungen dafür aufzeigen. Gibt es Ungereimtheiten? Sehen Sie spezifische und offensichtliche Merkmale in der CMD, die Sie nicht erklären können, die nicht Ihren Erwartungen entsprechen?

Hinweis: *Aufgrund der Plots und Bilder passt das Protokoll möglicherweise nicht in einen E-Mail-Anhang. Sie können Ihr Protokoll auf das Universitäts-Cloud-System (BoxUP) hochladen oder alternativ auf den Auswerterecher des Praktikums ablegen und uns den Pfad schicken.*

Übersicht: Praktikum