fit_isochrones

May 21, 2020

1 Measuring distance to NGC 3201

Written by Evgenii N.

1.1 Prerequisite code

[1]:
```
# Import libraries that we will use later in this notebook
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import subprocess
import shutil
import re
from io import StringIO
from shutil import copyfile
from shutil import which
from ccdproc import CCDData
from photutils.aperture import CircularAperture
from astropy.visualization import ZScaleInterval, MinMaxInterval, ImageNormalize

# Make images non-blurry on high pixel density screens
%config InlineBackend.figure_format = 'retina'


def set_plot_style():
    """Set global style"""

    SMALL_SIZE = 13
    NORMAL_SIZE = 15
    LARGE_SIZE = 17

    # Title size
    plt.rcParams['axes.titlesize'] = LARGE_SIZE

    # Axes label size
```

```python
    plt.rcParams['axes.labelsize'] = NORMAL_SIZE

    # Tick label size
    plt.rcParams['xtick.labelsize'] = SMALL_SIZE
    plt.rcParams['ytick.labelsize'] = SMALL_SIZE

    # Legend text size
    plt.rcParams['legend.fontsize'] = SMALL_SIZE

    plt.rcParams['font.size'] = NORMAL_SIZE

    plt.rcParams['legend.fontsize'] = NORMAL_SIZE


    # Grid color
    plt.rcParams['grid.color'] = '#cccccc'

    # Define plot size
    plt.rcParams['figure.figsize'] = [12, 8]

    # Marker size
    plt.rcParams['lines.markersize'] = 10


def show_image(image, title, title_y_offset, apertures=None):
    """
    Display an image.

    Parameters
    ---------

    image: astropy.nddata.ccddata.CCDData
        A fits image to show.

    title: str
        Plot title.

    apertures: list of CircularAperture
        List of apertures to plot over the image, optional.

    title_y_offset: float
        The offset of the title position.
    """

    # Scale the image similar to 'zscale' mode in DS9.
    # This makes easier to spot things in the image.
    interval=ZScaleInterval()
```

```python
    vmin, vmax = interval.get_limits(image)
    norm = ImageNormalize(vmin=vmin, vmax=vmax)

    plt.imshow(image, cmap='gray', norm=norm)  # Set color map and pixel scaling
    plt.xlabel('x [pixel]')  # Set axis labels
    plt.ylabel('y [pixel]')
    plt.title(title, y=title_y_offset)  # Set image title
    plt.colorbar()  # Show color bar

    # Expand the plot to the edges
    plt.tight_layout()

    if apertures is not None:
        apertures.plot(color='#33ff33', lw=2, alpha=0.8)


set_plot_style()
```

1.2 Plot color-magnitude diagram

[2]:
```python
def save_plot(fig, plot_dir, file_name):
    """
    Save a plot to a file.

    Parameters
    ----------

    fig: matplotlib.figure.Figure
        Plot's figure

    plot_dir: str
        Directory where the plot file is placed.

    file_name: str
        Plot file name

    """

    if not os.path.exists(plot_dir):
        os.makedirs(plot_dir)

    image_path = os.path.join(plot_dir, file_name)

    plt.savefig(image_path, fig=fig, dpi=150, transparent=False)
```

```python
def make_cmd(data_path, blue_mag, red_mag,
             x_label, y_label, title, xlims, ylims,
             y_offset=0, x_offset=0, legend_label=None,
             title_offset=-0.15):
    """
    Make a plot of colour magnitude diagram.

    Parameters
    ----------

    data_path: str
        Path to the CSV file containing magnitudes for stars.

    blue_mag, red_mag: str
        Names of the column containing magnitudes for the bluer and redder␣
 ↪filters.

    x_label, y_label: str
        Axes labels.

    title: str
        Plot's title.

    xlims, ylims: (low, high)
        Limits for the axes.

    y_offset: float
        Offset that is added to the Y values of the observed data.

    legend_label: str
        Optional legend label.

    title_offset: float.
        A small negative number used to shift the title,
        so that it does not overlap with the plot. Adjusted manually.
    """

    # Read magnitudes and colors from CSV file
    df = pd.read_csv(data_path)

    # Create a figure and axis object
    fig, ax = plt.subplots(1, 1)

    # Drop rows with missing values
    df = df.dropna(subset=[blue_mag, red_mag])
```

```
    x_values = df[blue_mag] - df[red_mag]

    # Show plot
    ax.scatter(x_values + x_offset, df[red_mag] + y_offset, zorder=2,
               color="#0084ff40",
               edgecolor="#0084ff",
               label=legend_label)

    # Show grid
    ax.grid(zorder=-1)

    # Set plot labels
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)

    # Set axes limits
    # ------

    if xlims is not None:
        ax.set_xlim(xlims)

    if ylims is not None:
        ax.set_ylim(ylims)

    if title_offset is None:
        title_offset = -0.15

    ax.set_title(title, y=title_offset)  # Set image title

    # Invert y axis
    ax.invert_yaxis()

    return fig, ax



magnitudes_dir = "../060_find_magnitudes/data"
magnitudes_path = os.path.join(magnitudes_dir, "magnitudes.csv")

all_plot_settings = [
    dict(magnitudes=["B", "V"], xlims=(-0.5, 3), ylims=(11.5, 20)),
]

figure_number = 0

for plot_settings in all_plot_settings:
    axes_magnitudes = plot_settings['magnitudes']
```

```python
if 'xlims' in plot_settings:
    xlims = plot_settings['xlims']
else:
    xlims = None

if 'ylims' in plot_settings:
    ylims = plot_settings['ylims']
else:
    ylims = None

blue_mag = axes_magnitudes[0]
blue_mag_lowcase = blue_mag.lower()
red_mag = axes_magnitudes[1]
red_mag_lowcase = red_mag.lower()
figure_number += 1

title = (
    f"Figure {figure_number}: Colours and magnitudes "
    "of stars in the direction of NGC 3201 globular cluster."
)

fig, ax = make_cmd(data_path=magnitudes_path,
                   blue_mag=f"{blue_mag_lowcase}_mag",
                   red_mag=f"{red_mag_lowcase}_mag",
                   x_label=f"{blue_mag} - {red_mag} colour index",
                   y_label=f"{red_mag} apparent magnitude",
                   xlims=xlims, ylims=ylims,
                   title=title)

# Expand the plot to the edges
fig.tight_layout()

save_plot(fig=fig,
          file_name=f"cmd_{blue_mag_lowcase}_{red_mag_lowcase}.png",
          plot_dir="images")
```

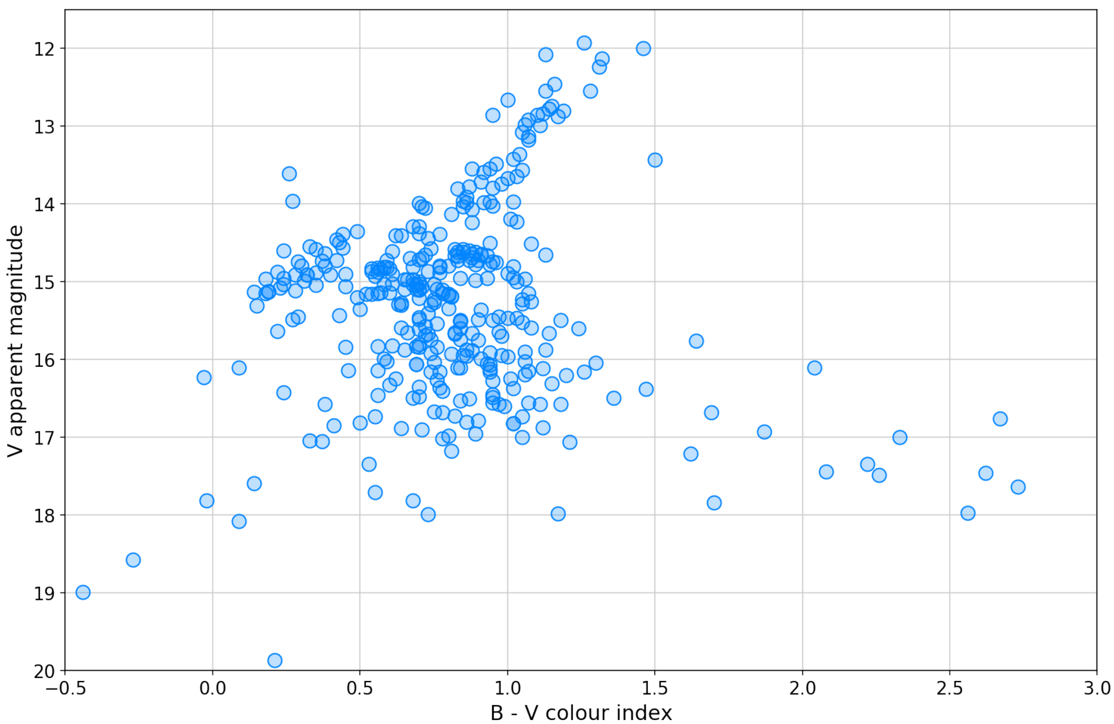

Figure 1: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

1.3 Does CMD look right?

Hmm, I don't see a main sequence on Fig. 1. In order to understand which stars am I looking at, I compare our plot with Layden et al. (2003) on Fig. 2.

- It looks like we don't have any meain sequence stars on our plot (right panel), so we can't use main sequence matching to find the distance to our globular cluster.

I posted Fig. 2 on the forum, here is response from Michael Brown:

> you cannot use main sequence fitting for the GC distance (and age), but you can use the red giant branch and isochrones in the same way. Indeed, you could have fun over plotting 100 Myr, 1 Gyr and 10 Gyr isochrones on your plot using the same distance modulus.
>
> There are tables of red giant colours and magnitudes available too, so you can do a cross check without relying on the isochrones alone.

Great! Let's try fitting Girardi isochrones and find distance and age.

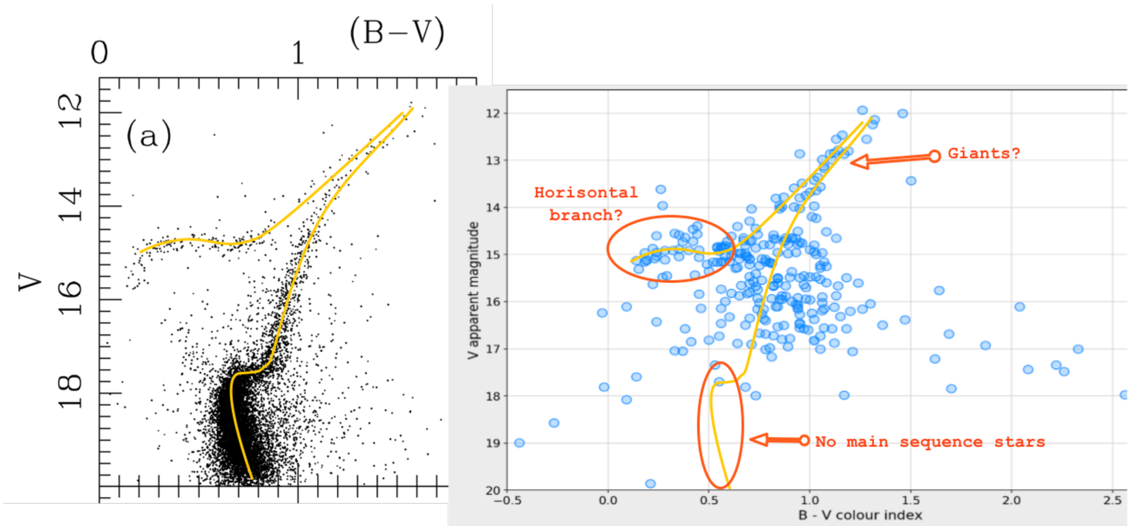

Figure 2: Comparison of color magnitude diagram: left is Fig. 2 from Layden et al. (2003), and the right is from our magnitude measurements. I drawn the evolutionary tracks by eye based on Layden. It appear that we don't have any main sequence stars in our data. However, I can see horizontal branch and giant stars, which is great.

1.4 Plot Girardi isochrone

I generate a Girardi isochrone:

- Go to http://stev.oapd.inaf.it/cgi-bin/cmd
- Set linear age (yr) to 12e9, based on Monty et al. 2018 estimate of 12.2 ± 0.5
- Set metallicity [M/H] to -1.5, based on Marino et al. (2019) estimate of -1.50 ± 0.02 (rms=0.07 dex).

Next, I plot the isochrone:

[3]:
```
def read_girardi_data(data_path):
    """
    Read Girardi isochrone data file,
    i.e. the file produced by http://stev.oapd.inaf.it/cgi-bin/cmd web site

    Parameters
    ----------


    data_path: str
        Path to data file

    Returns
    -------
    pandas.core.frame.DataFrame
        Data for Girardi iscochrone.
    """
```

```
    # Open text file
    with open(data_path) as file:
        file_text = file.read()
        # Remove comment character from header line
        file_text = file_text.replace('# Zini', "Zini")

        # Delete all comments
        file_text = re.sub(r'^#.*\n?', '', file_text, flags=re.MULTILINE)

        # Remove last line
        file_text = file_text[:file_text.rfind('\n',  0, len(file_text) - 1)]

    data = StringIO(file_text)

    # Read the data from text table,
    # using any whitespace characters as column separators
    return pd.read_table(data, delimiter='\s+')


def plot_isochrone(plot_dir, file_name, data_path):
    """
    Make a plot of a Girardi isochrone.

    Parameters
    ----------
    plot_dir: str
        Directory where the plot file is placed.

    file_name: str
        Plot file name.

    data_path: str
        Path to the text file output from http://stev.oapd.inaf.it/cgi-bin/cmd

    """

    df = read_girardi_data(data_path)

    # Create a figure and axis object
    fig, ax = plt.subplots(1, 1)

    # Show plot
    ax.plot(df["Bmag"] - df["Vmag"], df["Vmag"], zorder=2, c='red')

    # Set axes limits
    ax.set_xlim(-0.5, 2)
```

```
    ax.set_ylim(-4, 8)

    # Show grid
    ax.grid(zorder=-1)

    # Set plot labels
    ax.set_xlabel("B - V colour index")
    ax.set_ylabel("V absolute magnitude")

    # Invert y axis
    ax.invert_yaxis()

    # Expand the plot to the edges
    fig.tight_layout()

    save_plot(fig=fig, file_name=file_name, plot_dir=plot_dir)


plot_isochrone(plot_dir="images", file_name="girardi_12gyr.png",
               data_path="data/girardi/-1.5/12.00_gyr.txt")
```

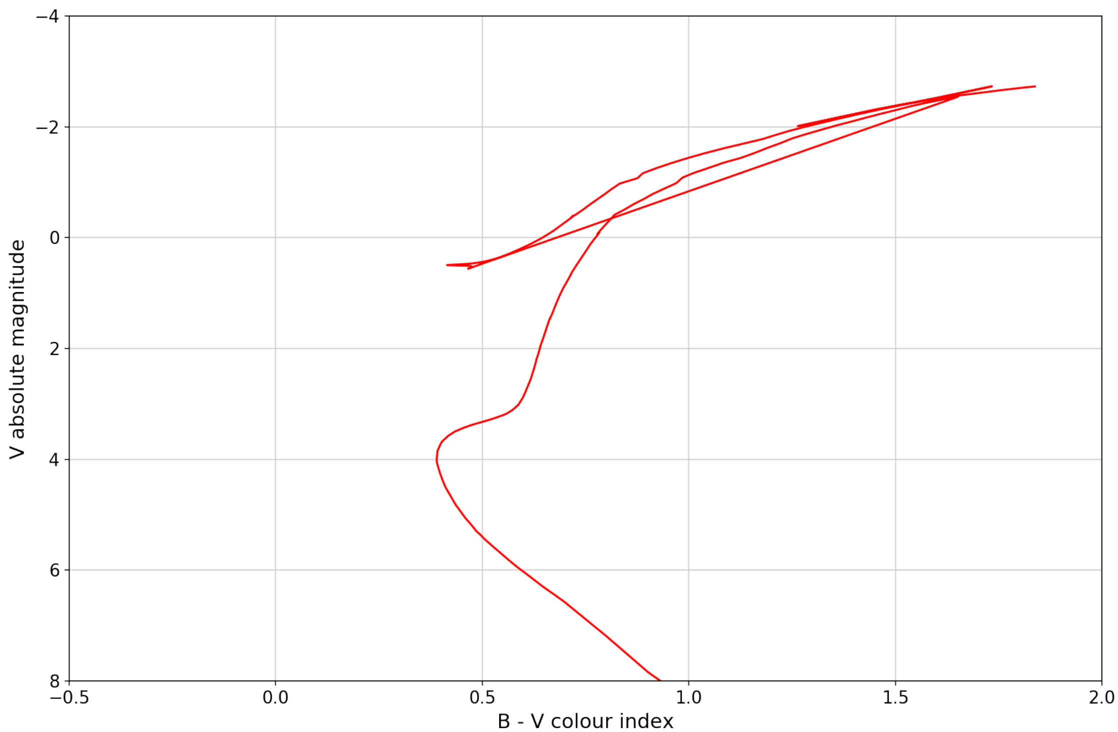

Figure 3: Girardi isochrone, 12 Gyr, [M/H]=-1.5.

[4]:
```
figure_number = 3
```

1.5 Fitting data to Girardi isochrone

First, I shift our measurements from Fig. 1 vertically (i.e. shift the V values) in order to match Girardi isochrone.

- I manually choose the amount of shift, which happens to be about 14.5 so that the observed giant branch is best matched with that from Girardi isochrone,
- This number is $(m-M)_V$, aka apparent visual distance modulus.

Next, I shift the measurements from Fig. 1 horizontally (i.e. shift the B-V values). This is done to compensate for reddening.

1.5.1 What on Earth is "reddening"?

Well, it turns out, the light is sometimes blocked by stuff and blue light is blocked more than red light. For example, the sunsets are red because there is a lot of air the light goes through when the sun is on the horizon, and this air blocks blue light more than red light (why? I don't know). Similarly, the blue light from NGC 3201 is blocked more than red light, so the stars we observed are more red than they really are. This is called "reddening".

1.5.2 How do I correct for reddening?

I need to make our measurements from Fig. 1 bluer, so I need to subtract some number from B-V values. This reddening number is $E(B-V) = 0.25 \pm 0.01$, it was calculated by Monty et al. 2018. So what I do here is to subtract 0.25 from all B-V values on Fig. 1.

1.6 Calculating distance

The point of all this trouble is to calculate the distance to NGC 3201. I can use equation

$$(m-M)_0 = 5\log_{10}\left(\frac{d}{10\text{ pc}}\right) \qquad (1)$$

to calculate distance d in parsecs. Notice that here $(m-M)_0$ is the *true distance modulus*. This is the distance modulus we would calculate if the light was not blocked by dust. To calculate $(m-M)_0$ I subtract extinction number A_V from my $(m-M)_V = 14.5$:

$$(m-M)_0 = (m-M)_V - A_V. \qquad (2)$$

Here, I calculate A_V using the reddening value $E(B-V) = 0.25$, multiplied by $R_V = 3.1$ O'Donnell (1994):

$$A_V = R_V E(B-V) = 3.1 * 0.25 = 0.775.$$

Why? I have no idea. Substituting $A_V = 0.775$ and $(m-M)_V = 14.5$ into Eq. 2 gives:

$$(m - M)_0 = (m - M)_V - A_V = 14.5 - 0.775 = 13.725.$$

Finally, I solve Eq. 1 for distance d, substitute $(m - M)_0$ calculate the distance I wanted:

$$d = (10)\left(10^{13.725/5}\right) \approx 5500 \text{ pc}.$$

I am done!

[5]:
```
def calculate_distance(apparent_distance_modulus, reddening, r_v=3.1):
    """
    Calculate distance given apparent distance modulus.

    Parameters
    ----------

    apparent_distance_modulus: float
        Apparent visual distance modulus

    reddening: float
        Reddening, aka E(B - V).
        Extinction correction value of the (B-V) colour.

    r_v: float
        Total-to-selective extinction ratio value that is dependent upon the
        density of the interstellar medium. Default value 3.1 form O'Donnell␣
↪(1994)
        https://ui.adsabs.harvard.edu/abs/1994ApJ...422..158O/abstract
    """

    # Calculate true distance modulus
    true_distance_modulus = apparent_distance_modulus - r_v * reddening

    # Calculate the distance
    return 10 * 10**(true_distance_modulus / 5)


def plot_cmd_with_girardi(plot_dir, file_name,
                          magnitudes_path, girardi_path,
                          x_label, y_label, xlims, ylims,
                          title, title_offset,
                          distance_modulus, reddening, text, show=False):
    """
    Plot V vs B-V color magnitude diagram containing observations and Girardi␣
↪isochrone.

    Parameters
```

```
    ----------
    plot_dir: str
        Directory where the plot file is placed.

    file_name: str
        Plot file name.


    girardi_path: str
        Path to the text file output from http://stev.oapd.inaf.it/cgi-bin/cmd

    magnitudes_path: str
        Path to the CSV file containing magnitudes for stars.

    x_label, y_label: str
        Axes labels.

    xlims, ylims: (low, high)
        Limits for the axes.

    title: str
        Plot's title.

    text: str
        Text shown in a text box on the plot.

    title_offset: float.
        A small negative number used to shift the title,
        so that it does not overlap with the plot. Adjusted manually.

    distance_modulus: float
        Apparent visual distance modulus.
        Used for shifting the observed data along the Y-axis
        and calculating the distance.

    reddening: float
        Reddening, aka E(B - V).
        Extinction correction value of the (B-V) colour.

    show: bool
        If True, this image is shown in the Notebook.
    """

    fig, ax = make_cmd(data_path=magnitudes_path,
                       blue_mag=f"b_mag",
                       red_mag=f"v_mag",
                       x_label=x_label,
```

```
                        y_label=y_label,
                        xlims=xlims, ylims=ylims,
                        title=title,
                        title_offset=title_offset,
                        legend_label="Observations")

    df = read_girardi_data(girardi_path)

    # Plot Girardi isochrone
    # ------

    # Make colors redder
    x_values = df["Bmag"] - df["Vmag"] + reddening

    # Make magnitudes dimmer
    y_values = df["Vmag"] + distance_modulus

    ax.plot(x_values, y_values,
            zorder=2, c='red', label="Girardi isochrone")

    # Show text box
    text = plt.text(
        0.033, 0.1,
        text,
        horizontalalignment='left',
        verticalalignment='center',
        transform=ax.transAxes,
        bbox=dict(facecolor='white', boxstyle='round', alpha=0.8, edgecolor='0.
↪7'))

    # Expand the plot to the edges
    fig.tight_layout()

    # Show legend
    ax.legend(loc='upper left')

    # Save plot to file
    save_plot(fig=fig, file_name=file_name, plot_dir=plot_dir)

    if show:
        plt.show()

    # Close figure to free up memory
    plt.close(fig)


def plot_cmd_with_girardi_with_title(plot_dir, distance_modulus, reddening,
```

```
                                             figure_number, girardi_dir,␣
↪girardi_age_gyr,
                                             girardi_metallicity, r_v=3.1, show=False):
    """
    Make the plot of observations with Girardi isochrone,
    with full title, containing different parameters.

    Parameters
    ----------
    plot_dir: str
        Directory where the plot file is placed.

    distance_modulus: float
        Apparent visual distance modulus.
        Used for shifting the observed data along the Y-axis
        and calculating the distance.

    reddening: float
        Reddening, aka E(B - V).
        Extinction correction value of the (B-V) colour.

    file_name: str
        Plot file name.

    girardi_dir: str
        Path to where Girardi isochrones are located.

    girardi_age_gyr: float
        Are used to generate the Figardi isochrone, in Gyr. Example: 12.

    girardi_metallicity: float
        Metallicity used to generate Figardi isochrone [M/H]. Example: -1.

    r_v: float
        Total-to-selective extinction ratio value that is dependent upon the
        density of the interstellar medium. Default value 3.1 form O'Donnell␣
↪(1994)
        https://ui.adsabs.harvard.edu/abs/1994ApJ...422..158O/abstract

    show: bool
        If True, this image is shown in the Notebook.
    """

    distance = calculate_distance(apparent_distance_modulus=distance_modulus,
                                  reddening=reddening, r_v=r_v)

    # Round the distance
```

```
    distance = round(distance / 100) * 100

    # Set plot title
    title = (
        f"Figure {figure_number}: Fitting Girardi isochrones to observed stars "
        "in direction of NGC 3201"
    )

    # Set text shown in a box on the plot
    text = (
        f"Age: {girardi_age_gyr:.2g} Ga\n"
        f"[M/H]: {girardi_metallicity:.1f}\n"
        "$(m-M)_V: $"
        f"{distance_modulus}\n"
        f"Distance: {distance} pc"
    )

    girardi_path = os.path.join(girardi_dir, f'{girardi_metallicity:.1f}',
                                f'{girardi_age_gyr:.2f}_gyr.txt')

    plot_file_name = f'cmd_{distance_modulus:.2f}mag__{girardi_age_gyr:.2f}gyr.
↪png'

    print(plot_file_name)

    plot_cmd_with_girardi(plot_dir=plot_dir,
                          file_name=plot_file_name,
                          magnitudes_path=magnitudes_path,
                          xlims=(-0.5, 2), ylims=(10, 20),
                          x_label="B - V colour index",
                          y_label="V apperant magnitude",
                          distance_modulus=distance_modulus,
                          reddening=reddening,
                          girardi_path=girardi_path,
                          title=title,
                          title_offset=None,
                          text=text,
                          show=show)

def delete_dir_if_exists(dir):
    """
    Creates a directory if exists exist.

    Parameters
    ----------
    dir : str
        Directory path, can be nested directories.
```

```python
    """
    if os.path.exists(dir):
        shutil.rmtree(dir)


def create_dir(dir):
    """
    Creates a directory if it does not exist.

    Parameters
    ----------
    dir : str
        Directory path, can be nested directories.

    """

    if not os.path.exists(dir):
        os.makedirs(dir)


def is_tool(name):
    """Check whether `name` is on PATH and marked as executable."""

    return which(name) is not None


def make_movie(plot_dir, movie_dir, out_file):
    """
    Creates a movie .mp4 file from individual images.

    Parameters
    ----------
    plot_dir : str
        Path to directory containing individual images for the movie.

    movie_dir : str
        The output directory of the movie.

    out_file: str
        Name of the output movie file.
    """

    # Temporary movie path
    src_movie = os.path.join(plot_dir, out_file)

    # Remove temporary movie if exists
```

```
    if os.path.exists(src_movie):
        os.remove(src_movie)

    # Check if ffmpeg is installed
    if not is_tool("ffmpeg"):
        raise RuntimeError("ERROR: ffmpeg is not installed. Install ffmpeg␣
↪program.")

    # Run ffmpeg to make the movie
    # ----------

    command = (f"ffmpeg -framerate 1 -pattern_type glob -i '*.png' "
               f"-c:v libx264 -pix_fmt yuv420p {out_file}")

    subprocess.call(command, cwd=plot_dir, shell=True)

    # Copy movie to output directory
    # ----------

    dest_movie = os.path.join(movie_dir, out_file)
    create_dir(movie_dir)

    if os.path.exists(dest_movie):
        os.remove(dest_movie)

    copyfile(src_movie, dest_movie)
    print(f"Movie saved to {dest_movie}")

    # Remove temporary movie file
    # ------
    os.remove(src_movie)
```

[6]:
```
# Directory with Girardi isochrones
girardi_dir = "data/girardi"

# Directory where temporary images for fitting are created
fit_images_dir = "images/fitting"

# Reddening, aka E(B - V)
# Value from Monty et al. 2018, https://arxiv.org/abs/1808.05271
reddening = 0.25

# Fit Girardi isochrone for observations.
# Create a movie of the fits for each Girardi metallicity value [M/H]
for girardi_metallicity in [-1, -1.5, -2]:
    # Remove fit directory if exists
    if os.path.exists(fit_images_dir):
```

```
        shutil.rmtree(fit_images_dir)

    # Create plot images
    # ----------

    print(f"\nMaking plots for [M/H]={girardi_metallicity:.1f}")

    # Iterate over distance moduli
    for distance_modulus in [13, 13.5, 14, 14.3, 14.6]:
        # Iterate over Girardi ages (Gyr)
        for girardi_age_gyr in range(10, 17):
            figure_number += 1

            plot_cmd_with_girardi_with_title(plot_dir=fit_images_dir,
                                             distance_modulus=distance_modulus,
                                             reddening=reddening,
                                             figure_number=figure_number,
                                             girardi_dir=girardi_dir,
                                             girardi_age_gyr=girardi_age_gyr,
                                             ␣
↪girardi_metallicity=girardi_metallicity)


    # Make a movie from plot images
    print("Making movie...")
    movie_file_name = f"cmd_fit_{girardi_metallicity:.1f}.mp4"
    make_movie(plot_dir=fit_images_dir, movie_dir="movies",␣
↪out_file=movie_file_name)


# Cleanup: remove fit directory
if os.path.exists(fit_images_dir):
    shutil.rmtree(fit_images_dir)

print("We are done!")
```

```
Making plots for [M/H]=-1.0
cmd_13.00mag__10.00gyr.png
cmd_13.00mag__11.00gyr.png
cmd_13.00mag__12.00gyr.png
cmd_13.00mag__13.00gyr.png
cmd_13.00mag__14.00gyr.png
cmd_13.00mag__15.00gyr.png
cmd_13.00mag__16.00gyr.png
cmd_13.50mag__10.00gyr.png
cmd_13.50mag__11.00gyr.png
```

```
cmd_13.50mag__12.00gyr.png
cmd_13.50mag__13.00gyr.png
cmd_13.50mag__14.00gyr.png
cmd_13.50mag__15.00gyr.png
cmd_13.50mag__16.00gyr.png
cmd_14.00mag__10.00gyr.png
cmd_14.00mag__11.00gyr.png
cmd_14.00mag__12.00gyr.png
cmd_14.00mag__13.00gyr.png
cmd_14.00mag__14.00gyr.png
cmd_14.00mag__15.00gyr.png
cmd_14.00mag__16.00gyr.png
cmd_14.30mag__10.00gyr.png
cmd_14.30mag__11.00gyr.png
cmd_14.30mag__12.00gyr.png
cmd_14.30mag__13.00gyr.png
cmd_14.30mag__14.00gyr.png
cmd_14.30mag__15.00gyr.png
cmd_14.30mag__16.00gyr.png
cmd_14.60mag__10.00gyr.png
cmd_14.60mag__11.00gyr.png
cmd_14.60mag__12.00gyr.png
cmd_14.60mag__13.00gyr.png
cmd_14.60mag__14.00gyr.png
cmd_14.60mag__15.00gyr.png
cmd_14.60mag__16.00gyr.png
Making movie…
Movie saved to movies/cmd_fit_-1.0.mp4

Making plots for [M/H]=-1.5
cmd_13.00mag__10.00gyr.png
cmd_13.00mag__11.00gyr.png
cmd_13.00mag__12.00gyr.png
cmd_13.00mag__13.00gyr.png
cmd_13.00mag__14.00gyr.png
cmd_13.00mag__15.00gyr.png
cmd_13.00mag__16.00gyr.png
cmd_13.50mag__10.00gyr.png
cmd_13.50mag__11.00gyr.png
cmd_13.50mag__12.00gyr.png
cmd_13.50mag__13.00gyr.png
cmd_13.50mag__14.00gyr.png
cmd_13.50mag__15.00gyr.png
cmd_13.50mag__16.00gyr.png
cmd_14.00mag__10.00gyr.png
cmd_14.00mag__11.00gyr.png
cmd_14.00mag__12.00gyr.png
cmd_14.00mag__13.00gyr.png
```

```
cmd_14.00mag__14.00gyr.png
cmd_14.00mag__15.00gyr.png
cmd_14.00mag__16.00gyr.png
cmd_14.30mag__10.00gyr.png
cmd_14.30mag__11.00gyr.png
cmd_14.30mag__12.00gyr.png
cmd_14.30mag__13.00gyr.png
cmd_14.30mag__14.00gyr.png
cmd_14.30mag__15.00gyr.png
cmd_14.30mag__16.00gyr.png
cmd_14.60mag__10.00gyr.png
cmd_14.60mag__11.00gyr.png
cmd_14.60mag__12.00gyr.png
cmd_14.60mag__13.00gyr.png
cmd_14.60mag__14.00gyr.png
cmd_14.60mag__15.00gyr.png
cmd_14.60mag__16.00gyr.png
Making movie…
Movie saved to movies/cmd_fit_-1.5.mp4

Making plots for [M/H]=-2.0
cmd_13.00mag__10.00gyr.png
cmd_13.00mag__11.00gyr.png
cmd_13.00mag__12.00gyr.png
cmd_13.00mag__13.00gyr.png
cmd_13.00mag__14.00gyr.png
cmd_13.00mag__15.00gyr.png
cmd_13.00mag__16.00gyr.png
cmd_13.50mag__10.00gyr.png
cmd_13.50mag__11.00gyr.png
cmd_13.50mag__12.00gyr.png
cmd_13.50mag__13.00gyr.png
cmd_13.50mag__14.00gyr.png
cmd_13.50mag__15.00gyr.png
cmd_13.50mag__16.00gyr.png
cmd_14.00mag__10.00gyr.png
cmd_14.00mag__11.00gyr.png
cmd_14.00mag__12.00gyr.png
cmd_14.00mag__13.00gyr.png
cmd_14.00mag__14.00gyr.png
cmd_14.00mag__15.00gyr.png
cmd_14.00mag__16.00gyr.png
cmd_14.30mag__10.00gyr.png
cmd_14.30mag__11.00gyr.png
cmd_14.30mag__12.00gyr.png
cmd_14.30mag__13.00gyr.png
cmd_14.30mag__14.00gyr.png
cmd_14.30mag__15.00gyr.png
```

```
cmd_14.30mag__16.00gyr.png
cmd_14.60mag__10.00gyr.png
cmd_14.60mag__11.00gyr.png
cmd_14.60mag__12.00gyr.png
cmd_14.60mag__13.00gyr.png
cmd_14.60mag__14.00gyr.png
cmd_14.60mag__15.00gyr.png
cmd_14.60mag__16.00gyr.png
Making movie…
Movie saved to movies/cmd_fit_-2.0.mp4
We are done!
```

1.7 Fitting videos

Watch the videos of all the plots I made above:

- [M/H]=-1.0: https://youtu.be/d5S5QkF6LlY
- [M/H]=-1.5: https://youtu.be/DgaelLLoA6Y
- [M/H]=-2.0: https://youtu.be/tVUJav2DzSo

1.8 Best Girardi fits

I've selected the best fits, shown on Figures 109-113. Parameters are:

- For [M/H]=-1.5, distance 5100 pc, age 16 Ga.
- For [M/H]=-2.0, distances 4400-5100 pc with ages 15-16 Ga.

[7]:
```
fit_images_dir = "images/good_fits"

good_fits = [
    dict(distance_modulus=14.3, age=16, metallicity=-1.5),
    dict(distance_modulus=14, age=15, metallicity=-2),
    dict(distance_modulus=14, age=16, metallicity=-2),
    dict(distance_modulus=14.3, age=15, metallicity=-2),
    dict(distance_modulus=14.3, age=16, metallicity=-2),
]

for good_fit in good_fits:
    figure_number += 1

    plot_cmd_with_girardi_with_title(plot_dir=fit_images_dir,
                                                          ␣
 ↪distance_modulus=good_fit['distance_modulus'],
                                     reddening=reddening,
                                     figure_number=figure_number,
                                     girardi_dir=girardi_dir,
```

```
                                    girardi_age_gyr=good_fit['age'],
                                                                   ␣
↪girardi_metallicity=good_fit['metallicity'],
                                    show=True)
```

cmd_14.30mag__16.00gyr.png

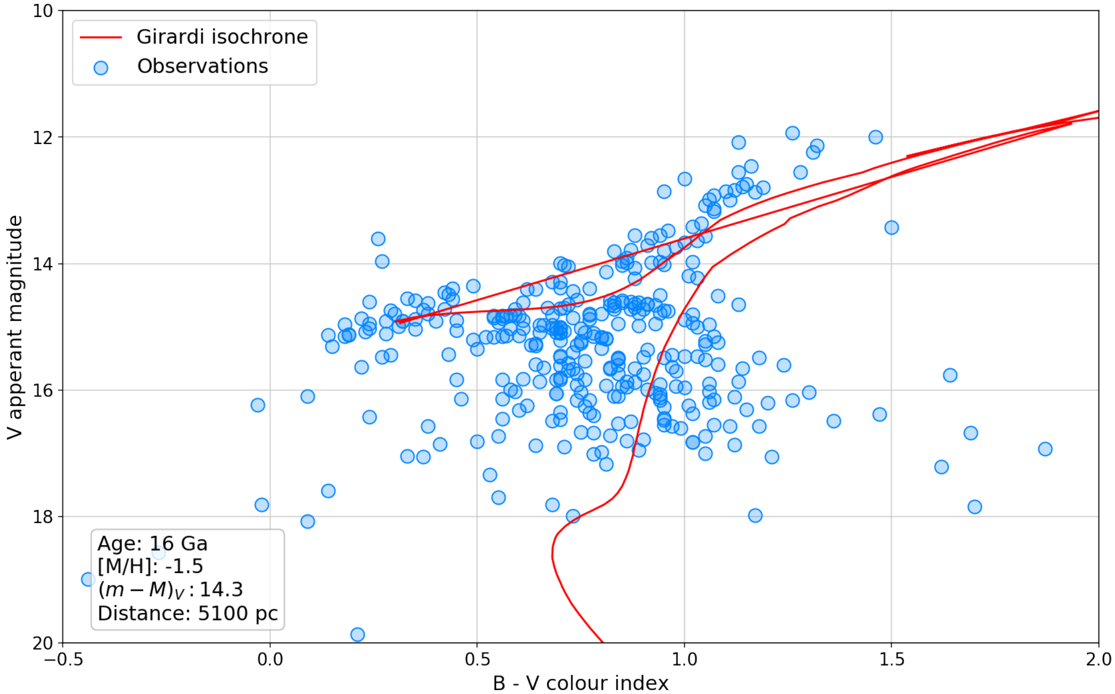

Figure 109: Fitting Girardi isochrones to observed stars in direction of NGC 3201

cmd_14.00mag__15.00gyr.png

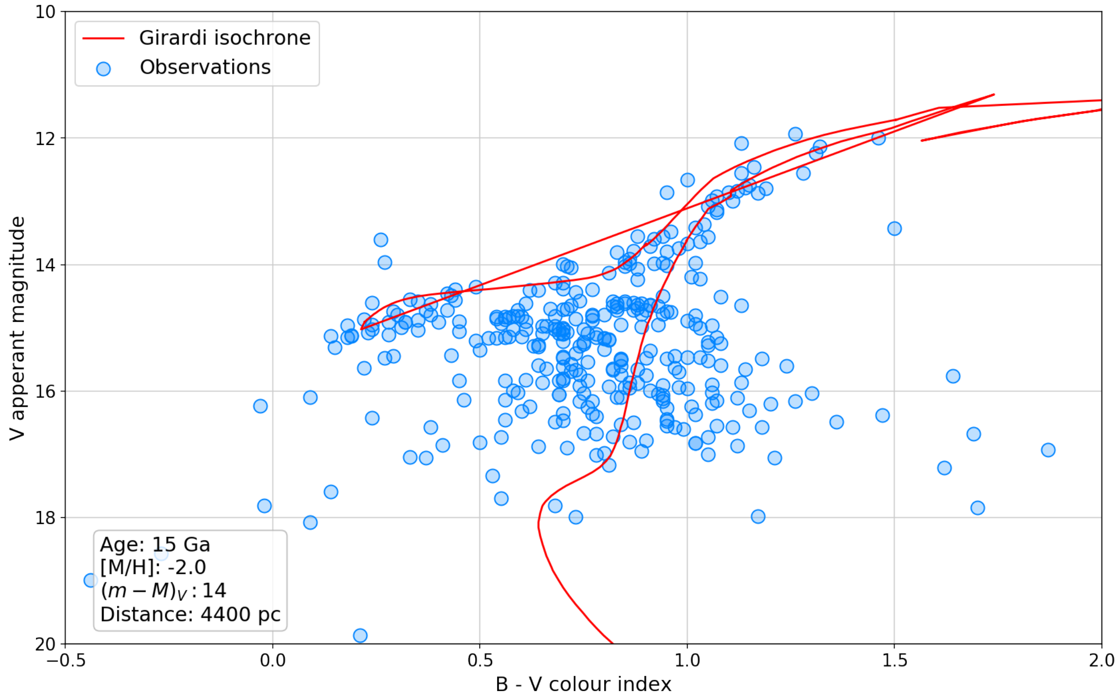

Figure 110: Fitting Girardi isochrones to observed stars in direction of NGC 3201

cmd_14.00mag__16.00gyr.png

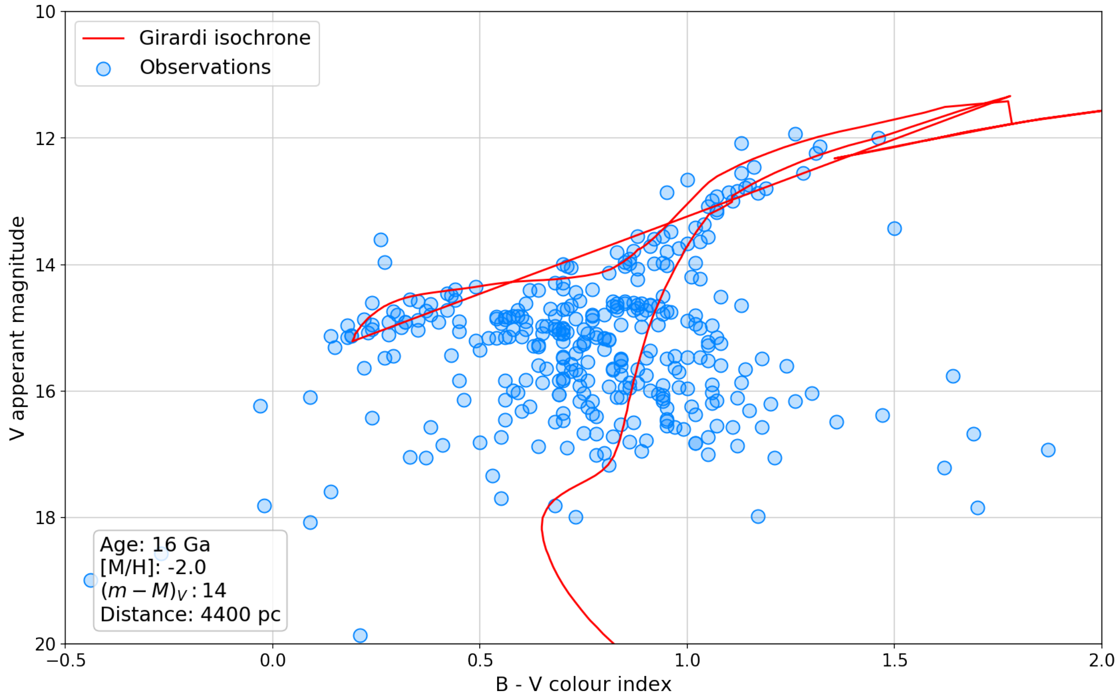

Figure 111: Fitting Girardi isochrones to observed stars in direction of NGC 3201

cmd_14.30mag__15.00gyr.png

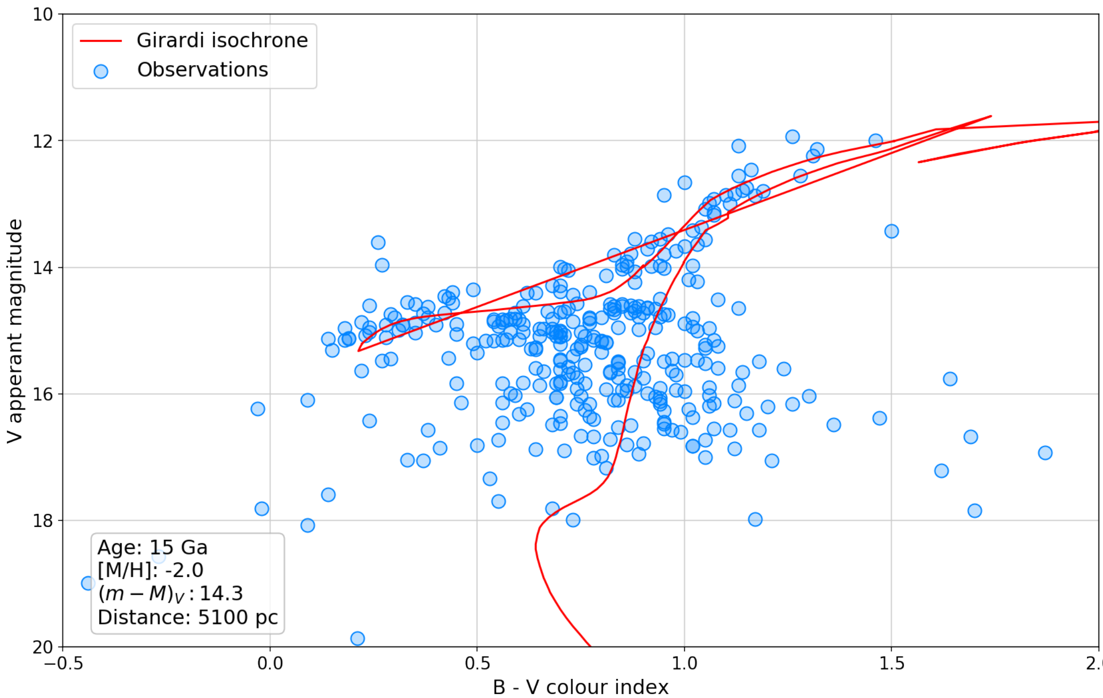

Figure 112: Fitting Girardi isochrones to observed stars in direction of NGC 3201

cmd_14.30mag__16.00gyr.png

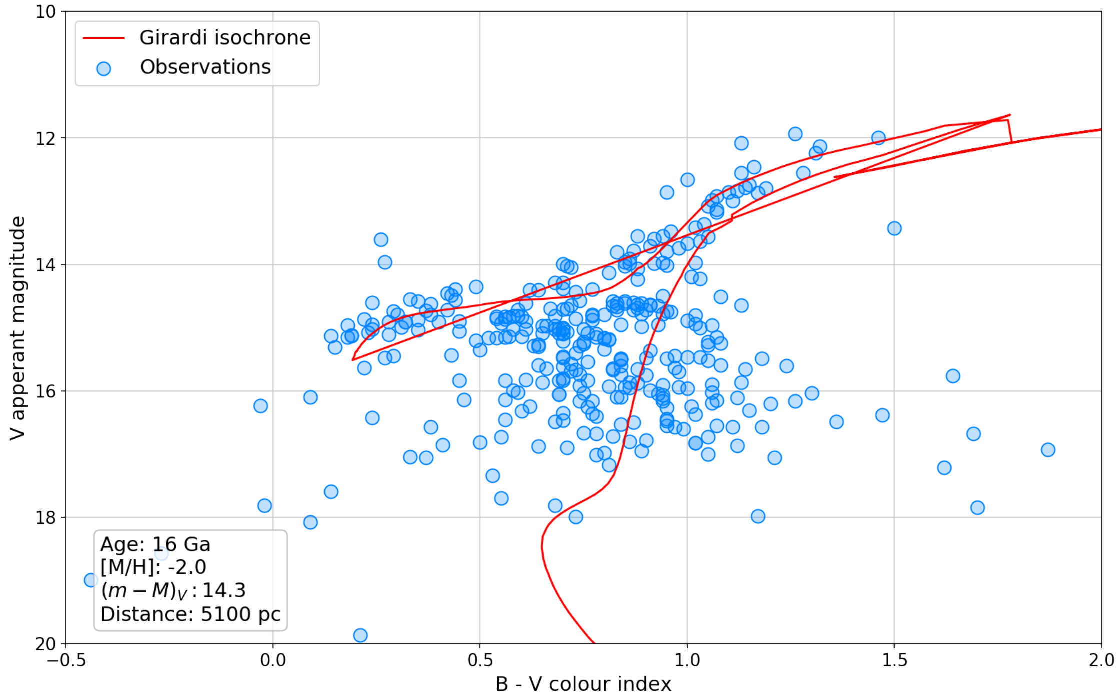

Figure 113: Fitting Girardi isochrones to observed stars in direction of NGC 3201

1.9 Comparing magnitudes of stars with Simbad

Michael Brown asked on the forum:

> How does your photometry of the brightest stars compare with Simbad/Aladin?

I want to compare B and V magnitudes that I measure with ones from SIMBAD:

- I manually select 20+ that comver range of magnitudes from 14 to 17.
- Make sure those stars are not reference stars.
- Write down x,y coordinate of the stars in `data/star_check.csv` file.
- Locate those stars on AladinLite web site, and write down their SIMBAD B and V magnitudes.

[8]:
```
def plot_stars(image, positions, star_numbers, aperture_radius,
               reference_stars, title, title_y_offset):
    """
    Plot the image and show positions of the stars on it.

    Parameters
    ----------

    image: astropy.nddata.ccddata.CCDData
```

```
        An image to the stars in.

    positions: list of (x, y)
        Position of all stars in the image.

    star_numbers: list of int
        Star numbers.

    aperture_radius: flost
        Radius of the star's aperture.

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions of reference stars.
    """

    apertures = CircularAperture(positions, r=aperture_radius)
    show_image(image=image, apertures=apertures, title=title,␣
↪title_y_offset=title_y_offset)

    reference_positions = [
        (star[0], star[1])
        for star in reference_stars[["non_photometric_x", "non_photometric_y"]].
↪values
    ]

    reference_apertures = CircularAperture(reference_positions,␣
↪r=aperture_radius * 1.3)
    reference_apertures.plot(color='#ff7777', lw=3, alpha=1)


    for number, position in zip(star_numbers, positions):
        plt.text(x=position[0], y=position[1] - 22, s=str(number),␣
↪color="white",
                 horizontalalignment='center', fontsize=10)

    plt.gca().invert_xaxis()
    plt.gca().invert_yaxis()

    plt.show()


def load_check_stars(star_check_path, figure_number, aperture_radius):
    """
    Load the stars I want to verify and plot them

    Parameters
    ----------
```

```
    star_check_path: str
        Path to CSV file containing positions of the check stars.

    figure_number: int
        Figure number to be shown in plot caption.

    aperture_radius: float
        Radius of aperture that will be used for marking check stars in the␣
↪plot.

    Returns
    -------
    pandas.core.frame.DataFrame
        Table containing the check stars and their magnitudes.
    """

    # Load stars we want to check
    df_star_check = pd.read_csv(star_check_path, index_col="star_number")
    star_check_positions = df_star_check[['x', 'y']].values

    reference_stars_path = os.path.join("../060_find_magnitudes/data",␣
↪"reference_stars.csv")
    reference_stars = pd.read_csv(reference_stars_path, index_col="star_number")

    # Read V filter image
    # --------

    non_photometric_dir = "../050_scaling_and_combining/march_09_2018_stacked"
    filter_name = "B"

    # Set path to non-photometric image
    image_path = os.path.join(non_photometric_dir,
                              f'NGC_3201_{filter_name.lower()}_median_60.0s.
↪fits')

    # Read non-photometric image
    image = CCDData.read(image_path)

    figure_number += 1

    title = (
        f"Figure {figure_number}: Stars selected for verification (green␣
↪circles), {filter_name} filter. \n"
        "Red circles are the reference stars used for calculating magnitudes in␣
↪photometric image."
    )
```

```
    plot_stars(image=image, positions=star_check_positions,
               star_numbers=df_star_check.index.values,
               aperture_radius=aperture_radius,
               reference_stars=reference_stars, title=title, title_y_offset=-0.
 ↪25)

    # Load all stars
    magnitudes_dir = "../060_find_magnitudes/data"
    magnitudes_path = os.path.join(magnitudes_dir, "magnitudes.csv")
    df_all_stars = pd.read_csv(magnitudes_path)

    # Drop rows with missing values
    df_star_check = df_star_check.dropna(subset=["b_mag_simbad",␣
 ↪"v_mag_simbad"])

    # Join two tables
    df_star_check['star_no'] = df_star_check.index
    df_joined = df_star_check.merge(df_all_stars, how='inner',␣
 ↪left_on=['x','y'],right_on=['x','y'])

    return df_joined

star_check_path = os.path.join("data", "star_check.csv")
figure_number += 1

df_check_stars = load_check_stars(star_check_path=star_check_path,
                                  figure_number=figure_number,␣
 ↪aperture_radius=8)
```

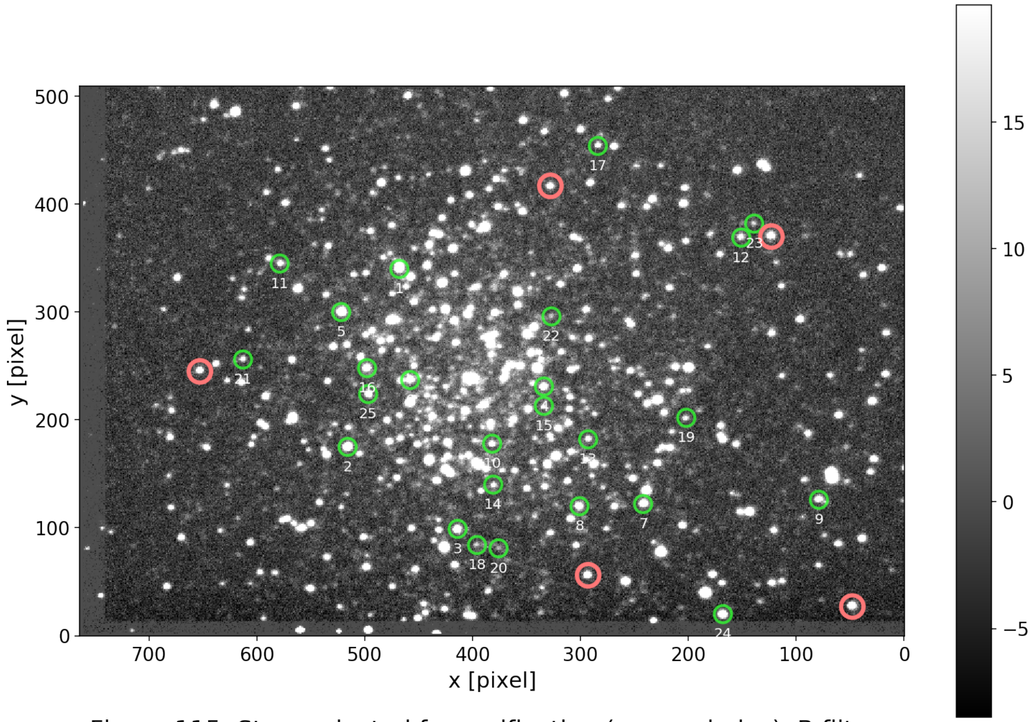

Figure 115: Stars selected for verification (green circles), B filter.
Red circles are the reference stars used for calculating magnitudes in photometric image.

Stars selected for verification are shown on Figure 115.

1.10 Plot our vs SIMBAD magnitudes

[9]:
```
def plot_simbad_vs_our_mag(df, filter_name, figure_number, star_label_offset=-0.
↪14):
    """
    Plot magnitudes from Simbad vs ours.

    Parameters
    ---------

    df: pandas.core.frame.DataFrame
        Table containing the magnitudes.

    filter_name: str
        Filter name: "V", "B" etc
    """

    filter_name = filter_name.lower()
    fig, ax = plt.subplots(1, 1, figsize=[10, 10])
```

```
    # Drop rows with empty magnitudes
    df = df.dropna(subset=[f"{filter_name}_mag"])

    # Show scatter plots
    x_values = df[f"{filter_name}_mag_simbad"]
    y_values = df[f"{filter_name}_mag"]
    ax.scatter(x_values, y_values, color="#0084ff40", edgecolor="#0084ff",␣
 ↪zorder=2)

    for b_simbad, b_mag, no in zip(x_values, y_values, df["star_no"]):
        ax.text(b_simbad, b_mag + star_label_offset, s=no,␣
 ↪horizontalalignment='center', fontsize=14,
               color="#0084ff")

    # Plot diagonal
    xlim = ax.get_xlim()
    ax.plot(xlim, xlim, linestyle='dashed', color="red", zorder=1)

    # Show grid
    ax.grid()

    # Set plot labels
    ax.set_xlabel(f"SIMBAD {filter_name.upper()} magnitude")
    ax.set_ylabel(f"This study {filter_name.upper()} magnitude")

    title = f"Figure {figure_number}: Compare measured {filter_name.upper()}␣
 ↪apparent magnitude with SIMBAD."
    ax.set_title(title, y=-0.12)  # Set image title
    ax.set_aspect('equal', adjustable='box')
    plt.tight_layout();


figure_number = 120

plot_simbad_vs_our_mag(df=df_check_stars, filter_name='B',␣
 ↪figure_number=figure_number)

figure_number += 1
plot_simbad_vs_our_mag(df=df_check_stars, filter_name='V',␣
 ↪figure_number=figure_number)

# Plot B-V
df_check_stars["b-v_mag"] = df_check_stars["b_mag"] - df_check_stars["v_mag"]
df_check_stars["b-v_mag_simbad"] = df_check_stars["b_mag_simbad"] -␣
 ↪df_check_stars["v_mag_simbad"]
```

```
figure_number += 1
plot_simbad_vs_our_mag(df=df_check_stars, filter_name='B-V',␣
↪figure_number=figure_number,
                        star_label_offset=-0.06)
```

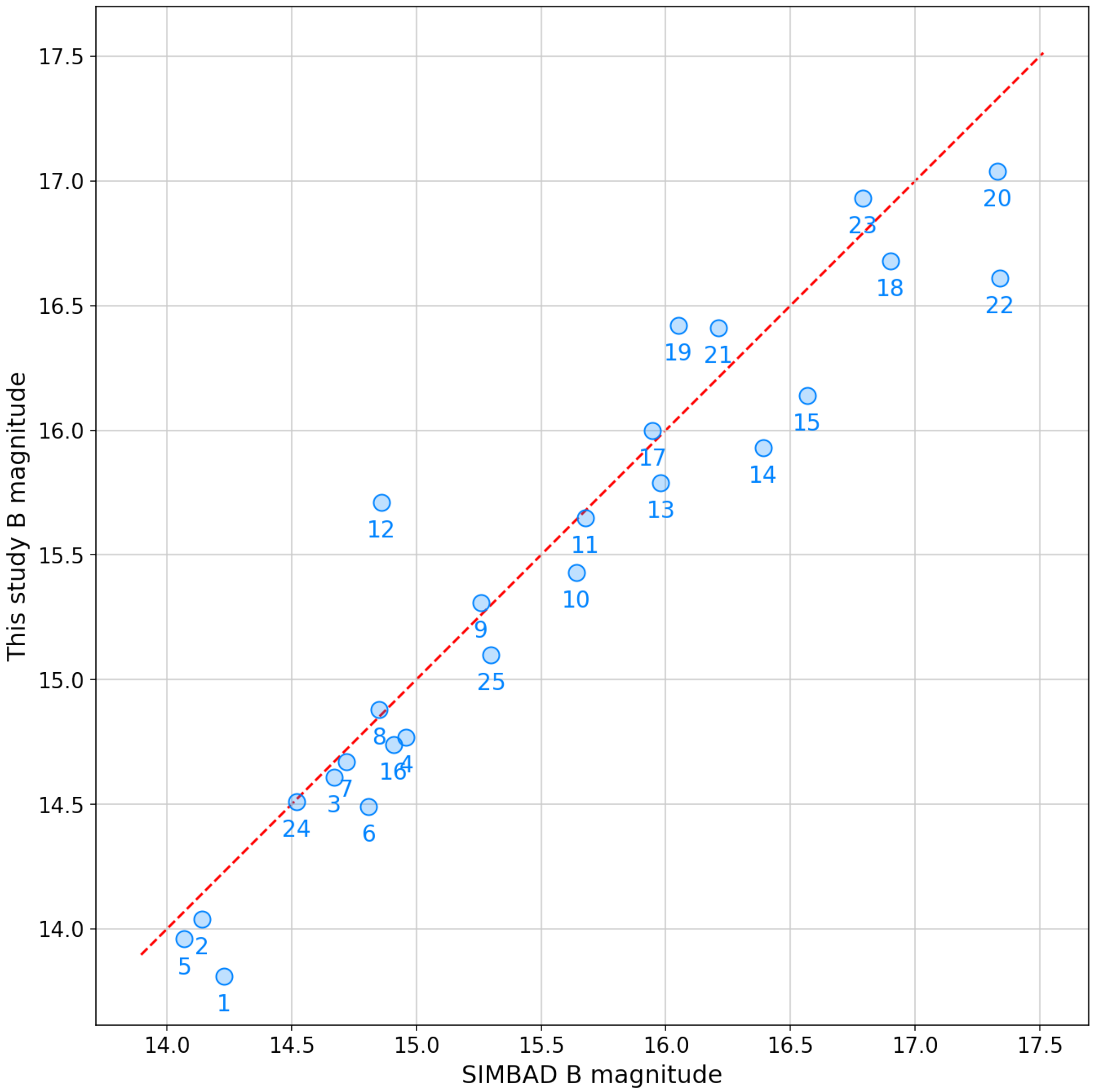

Figure 120: Compare measured B apparent magnitude with SIMBAD.

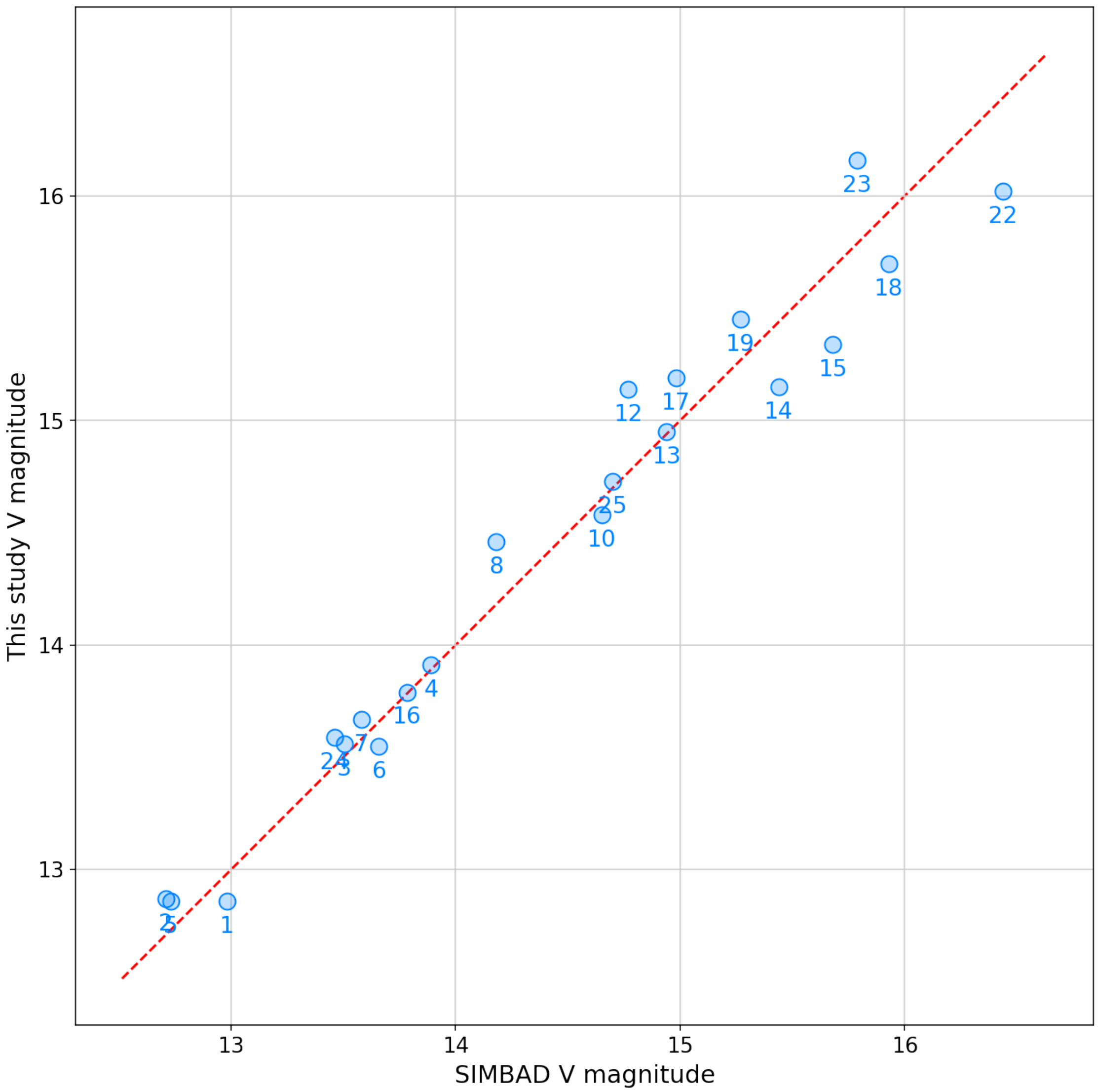

Figure 121: Compare measured V apparent magnitude with SIMBAD.

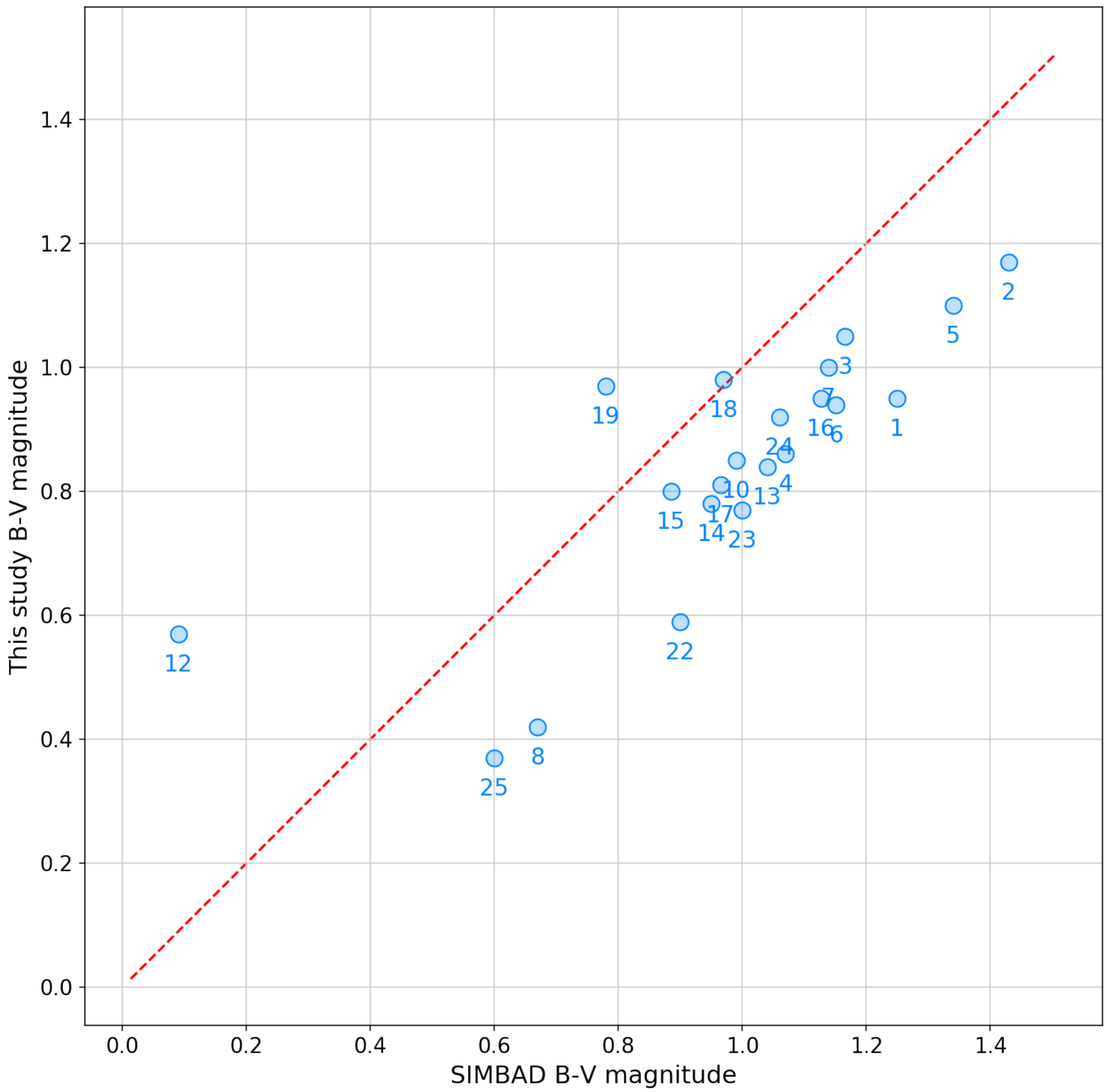

Figure 122: Compare measured B-V apparent magnitude with SIMBAD.

I can see from Figures 120 and 121 that my magnitudes differ from SIMBAD more for fainter stars. This makes sense, since singnal-to-noise is lower for fainter stars. Let's compare B-V.

1.11 Plot SIMBAD - our magnitudes

Let's print the different between SIMBAD and our magnitudes.

[10]:
```
def plot_difference_from_simbad(df, filter_name, figure_number, number_offset):
    """
    Plot magnitudes from Simbad vs ours.
```

```
    Parameters
    ----------

    df: pandas.core.frame.DataFrame
        Table containing the magnitudes.

    filter_name: str
        Filter name: "V", "B" etc.

    number_offset: float
        y-offset for the star label.
    """

    filter_name = filter_name.lower()
    fig, ax = plt.subplots(1, 1)

    # Drop rows with empty magnitudes
    df = df.dropna(subset=[f"{filter_name}_mag"])

    y_values = df[f"{filter_name}_mag_simbad"] - df[f"{filter_name}_mag"]

    # Show scatter plots
    ax.scatter(df[f"{filter_name}_mag_simbad"], y_values,
               color="#0084ff40",
               edgecolor="#0084ff", zorder=2)

    # Plot zero
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    ax.plot(xlim, (0, 0), linestyle='dashed', color="red", zorder=1)
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

    for x, y, no in zip(
        df[f"{filter_name}_mag_simbad"],
        y_values, df["star_no"]):
        ax.text(x, y + number_offset, s=no, horizontalalignment='center',
                fontsize=14, color="#0084ff")

    # Show grid
    ax.grid()

    # Set plot labels
    ax.set_xlabel(f"SIMBAD {filter_name.upper()} magnitude")
    ax.set_ylabel(f"SIMBAD - This study, {filter_name.upper()} magnitude")
```

```
    title = f"Figure {figure_number}: Compare {filter_name.upper()} measured apparent magnitude with SIMBAD."
    ax.set_title(title, y=-0.13)  # Set image title
    plt.tight_layout();


figure_number += 1
plot_difference_from_simbad(df=df_check_stars, filter_name='B',
                            figure_number=figure_number, number_offset=0.025)

figure_number += 1
plot_difference_from_simbad(df=df_check_stars, filter_name='V',
                            figure_number=figure_number, number_offset=0.015)

figure_number += 1
plot_difference_from_simbad(df=df_check_stars, filter_name='B-V',
                            figure_number=figure_number, number_offset=0.015)
```

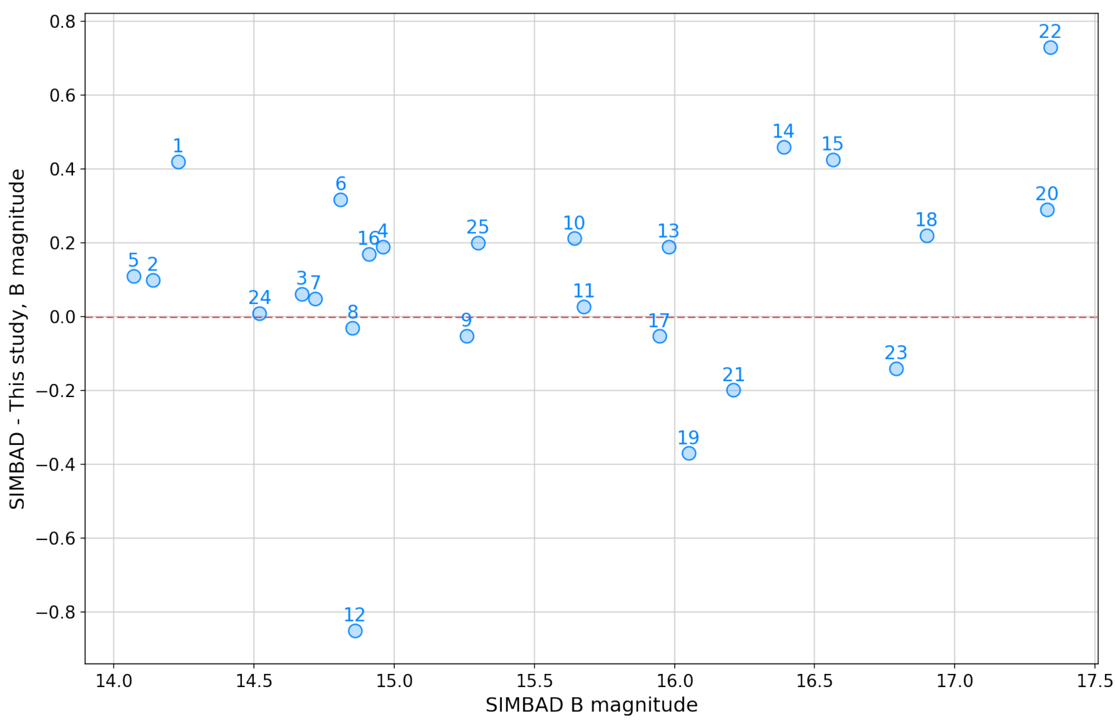

Figure 123: Compare B measured apparent magnitude with SIMBAD.

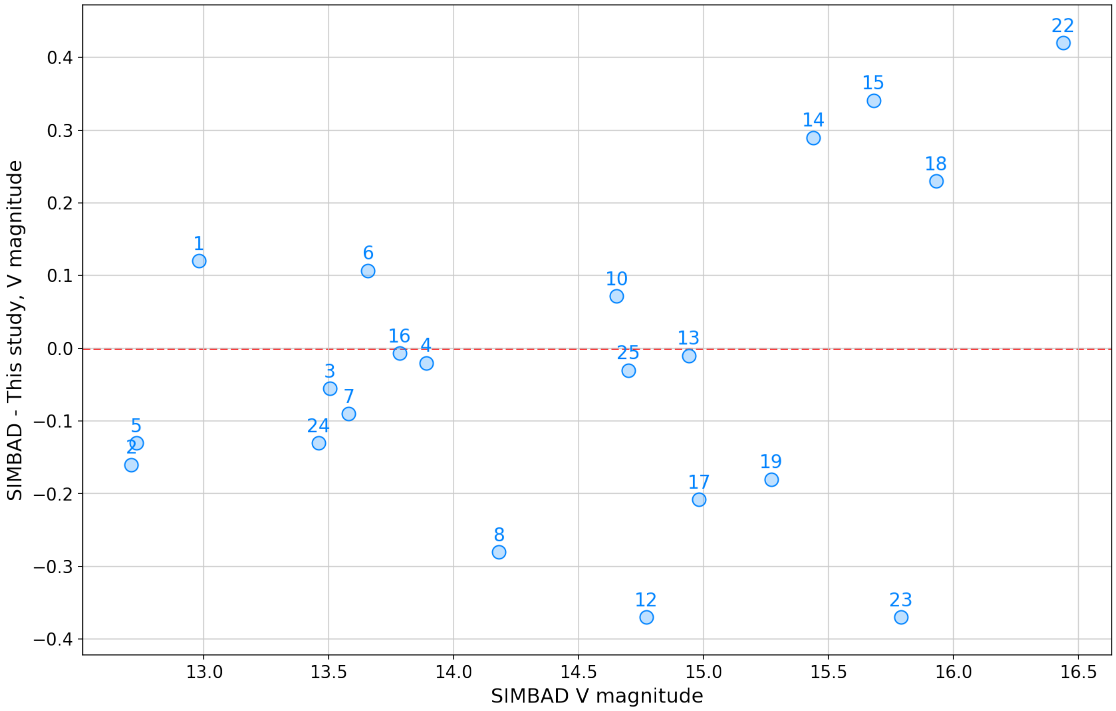

Figure 124: Compare V measured apparent magnitude with SIMBAD.

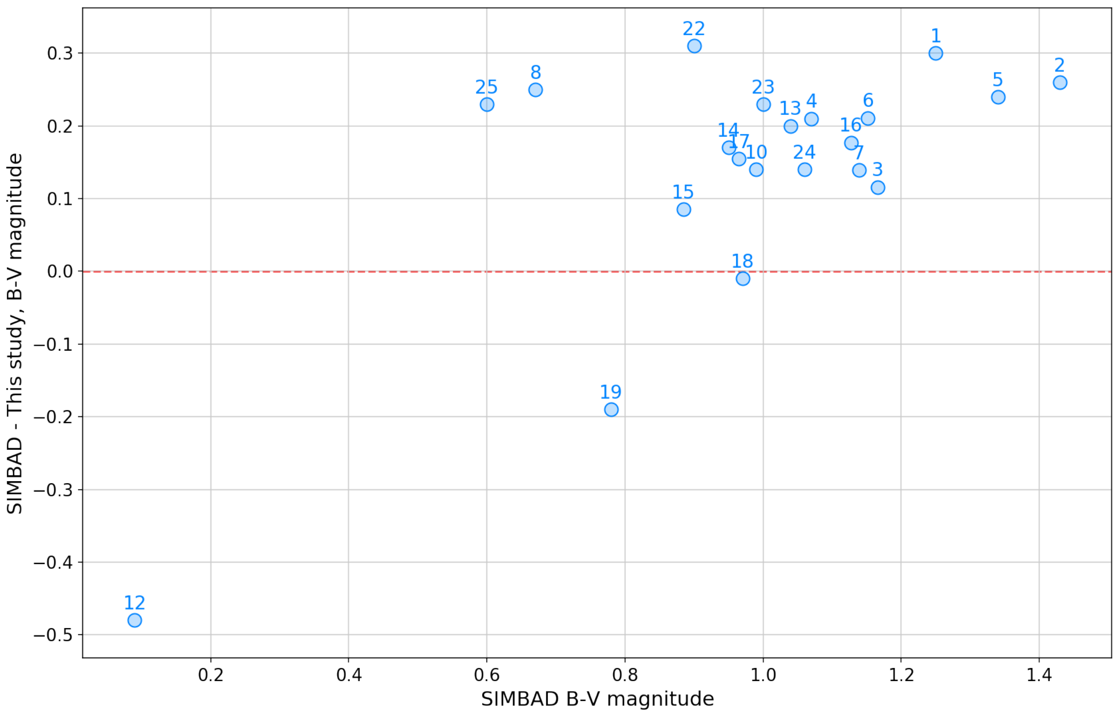

Figure 125: Compare B-V measured apparent magnitude with SIMBAD.

1.12 Make boxplots of SIMBAD - our magnitudes

[11]:
```
# The default colors used for box plots
BOXPLOT_COLORS = ['dodgerblue', 'lightgreen', 'white']
DEFAULT_COLOR = 'khaki'


def compare_boxplots(groups, title, figure_number, labels,␣
↪colors=BOXPLOT_COLORS):
    """
    Shows multiple box plots on one graph, in groups.

    Parameters
    ----------
    groups : list of list
        Each element of the list contains a list of values.

    title: str
        The main title of the plot.

    labels: list of str
        The x-axis labels for the box plots in each graph.

    colors: list of str
        A list of colors to use for the boxplots. The number of colors needs to␣
↪be the same as
        the number of paths in each element of paths array.
    """

    fig, ax = plt.subplots(figsize=[7, 9])

    if colors is None:
        colors = [DEFAULT_COLOR] * len(groups[0])

    bplot = ax.boxplot(groups, labels=labels, widths=0.5, patch_artist=True)

    for patch, color in zip(bplot['boxes'], colors):
        patch.set_facecolor(color)

    ax.grid()
    ax.set_title(title, y=-0.13)  # Set image title


def plot_difference_boxplot(df, figure_number, number_offset):
    """
    Plot magnitudes from Simbad vs ours.
```

```
    Parameters
    ----------

    df: pandas.core.frame.DataFrame
        Table containing the magnitudes.

    filter_name: str
        Filter name: "V", "B" etc.

    number_offset: float
        y-offset for the star label.
    """

    # Drop rows with empty magnitudes
    df_clean = df.dropna(subset=[f"v_mag"])
    v_values = df_clean[f"v_mag_simbad"] - df_clean[f"v_mag"]

    df_clean = df.dropna(subset=[f"b_mag"])
    b_values = df_clean[f"b_mag_simbad"] - df_clean[f"b_mag"]

    df_clean = df.dropna(subset=[f"b_mag", "v_mag"])
    b_minus_v_values = df_clean[f"b-v_mag_simbad"] - df_clean[f"b-v_mag"]

    title = f"Figure {figure_number}: Compare measured apparent magnitudes:
 ↪\nSIMBAD - this study."

    groups = [b_values.values, v_values.values, b_minus_v_values.values]

    compare_boxplots(groups=groups,
                     title=title,
                     figure_number=figure_number,
                     labels=["B", "V", "B-V"])


figure_number += 1

plot_difference_boxplot(df=df_check_stars, figure_number=figure_number,␣
 ↪number_offset=0.025)
```

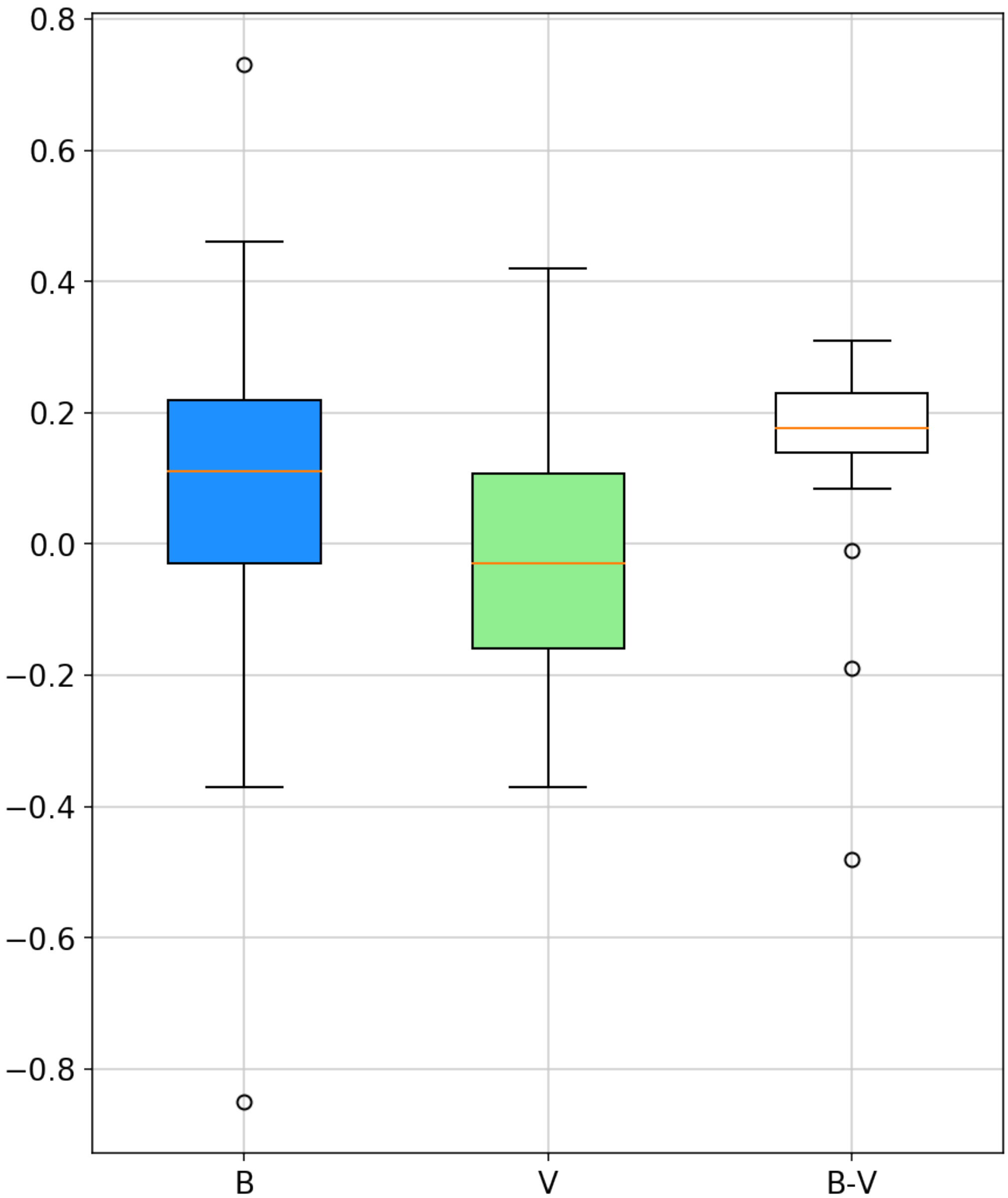

Figure 126: Compare measured apparent magnitudes: SIMBAD - this study.

1.13 Are amgnitudes any good?

- From Fig. 123, 126 I can see that our B magnitudes are systematically lower than SIMBAD's.
- V magnitudes are not biased (Fig. 124, 126).
- Our B-V is about 0.2 mag lower than SIMBAD's on average (Fig. 125 and 126).

[12]:
```
print("We are done!")
```

```
We are done!
```