find_magnitudes

May 21, 2020

1 Calculate magnitudes of the stars

Written by Evgenii N.

This code calculates the magnitudes of stars.

1.1 Prerequisite code

[1]:
```
# Import libraries that we will use later in this notebook
import os
import shutil
import ccdproc
import numpy as np
from astropy.visualization import ZScaleInterval, MinMaxInterval, ImageNormalize
from astropy import units as u
from astropy.stats import sigma_clipped_stats
from matplotlib.colors import LogNorm
from ccdproc import CCDData
import matplotlib.pyplot as plt
from photutils.aperture import CircularAperture, aperture_photometry
from photutils.centroids import centroid_2dg, centroid_com, centroid_1dg
from scipy.ndimage import shift
import pandas as pd
from tabulate import tabulate
from photutils import DAOStarFinder
from itertools import cycle
from cycler import cycler

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


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

    # Title size
    plt.rcParams['axes.titlesize'] = 17
```

```
    # Axes label size
    plt.rcParams['axes.labelsize'] = 15

    # Tick label size
    plt.rcParams['xtick.labelsize'] = 13
    plt.rcParams['ytick.labelsize'] = 13

    # Legend text size
    plt.rcParams['legend.fontsize'] = 13

    # 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, 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.
    """

    # Scale the image similar to 'zscale' mode in DS9.
    # This makes easier to spot things in the image.
    interval=ZScaleInterval()
    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=-0.2)  # 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=1.5, alpha=0.8)


def print_image_stats(image, title):
    """
    Print first pixel value, average and standard deviation for an image.

    Parameters
    ---------

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

    title: str
        Image name.
    """

    data = np.asarray(image)# Get numpy array for image data
    label_len = 10  # Length of the text label
    first_pixel = data[0, 0]  # First pixel
    average = np.mean(data)  # Average
    standard_deviation = np.std(data)   # Standard deviation


    # Print values
    # -------

    print(
        f'\n{title}',
        f"\n{'-' * len(title)}",
        f"\n{'Pixel:':<10}{first_pixel:>10.2f} ADU",
        f"\n{'Avg:':<10}{average:>10.2f} ADU",
        f"\n{'Std:':<10}{standard_deviation:>10.2f} ADU\n"
    )


def save_image(image, file_path):
    """
    Save image to disk. Overwrites the file if it already exist.
```

```
    Parameters
    ---------
    image: astropy.nddata.ccddata.CCDData
        Image to be saved

    file_path: str
        Path where the image is saved
    """

    # Delete the file if it already exists

    try:
        os.remove(file_path)
    except OSError:
        pass

    # Create directory
    # ------

    dirname = os.path.dirname(file_path)

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

    image.write(file_path)

set_plot_style()
```

1.2 Selecting reference stars

I have choosen the reference stars shown in Table 1. These are the stars that I will use to calculate marnitudes of all other stars in non-photometric night image (March 9) using the photometric callibrations (Eq. 1-4) and measurements from photometric night image (March 29). I have chosen the stars using the following method:

- In DS9, opened images for all filters from photometric night, located in code/050_scaling_and_combining/march_29_2018_stacked directory and well as code/040_shift/data/shifted/march_29_2018/NGC_3201_B_30.000secs_00000472.fit image.
- Chose Zoom > Invert Y in DS9.
- In DS9, opened images for all filters from non-photometric night, located in code/050_scaling_and_combining/march_09_2018_stacked directory.
- Chose Zoom > Invert X in DS9. Now all FITS images are in same orientation. The images also almost match orientation on Aladin Lite web site: our fits files are rotated about 3 degrees clockwise with respect to Aladin.

- First I use the B-filter photometric image and look at stars that are visible on it (meaning they have significant blue component).
- Then, I locate those stars on other FITS images and make sure they are not oversaturated, meaning that the radius is not larger than 10 pixels.
- I make sure there are no other bright stars closer than 20 pixels. Our science images are oversaturated and it is hard to distibuish stars in the center of the cluter. Because of this, I used stars far away from the center.

Table 1: X-y pixel coordinates of stars from photomoteric (March 29 2018) and non-photometric (March 9 2018) images. RA/DEC coordinates and apparent B, V magnitudes are taken from SIMBAD database using Aladin Lite web site. Uncertainties for magnitudes were missing for all stars in SIMBAD except for V magnitude of star 2 (uncertainty 0.002 mag).

#	x, y position, March 29 2018	x, y position, March 9 2018	RAJ2000 [deg]	DEJ2000 [deg]	B	V
1	738, 468	48, 27	154.3111238862	-46.4466457933	14.730	13.800
2	494, 439	293, 56	154.3751324274	-46.4435287544	15.146	14.021
3	664, 127	123, 370	154.3346246701	-46.3859927913	15.020	13.870
4	460, 80	328, 417	154.3882668536	-46.3793802270	15.560	14.770
5	135, 250	653, 245	154.4705311270	-46.4124240426	15.520	14.540

Locations of the reference stars in the images are shown on Fig 1.

Figure 1: Five reference stars from Table 1. Top: photometric image from March 29 2018, B-band, y-axis is inverted. Middle: non-photometric image, March 9 2018, I-band, x-axis is inverted. Bottom: view of the cluster from Aladin Lite selected stars are from SIMBAD database.

1.3 Photometric callibation

The following equations are photometric calibrations for March 9 2018 image, supplied to us by teachers:

$B = 19.237 - 2.5\log(f/t) - 0.330 * A \quad (1)$

$V = 19.696 - 2.5\log(f/t) - 0.210 * A \quad (2)$

$R = 19.679 - 2.5\log(f/t) - 0.119 * A \quad (3)$

$I = 19.079 - 2.5\log(f/t) - 0.134 * A \quad (4)$,

where

- A is the airmass number, taken from AIRMASS value from header of the FITS file,

- f is the star's flux we measure in the image, in ADU,
- t is exposure time of the frame, in seconds.

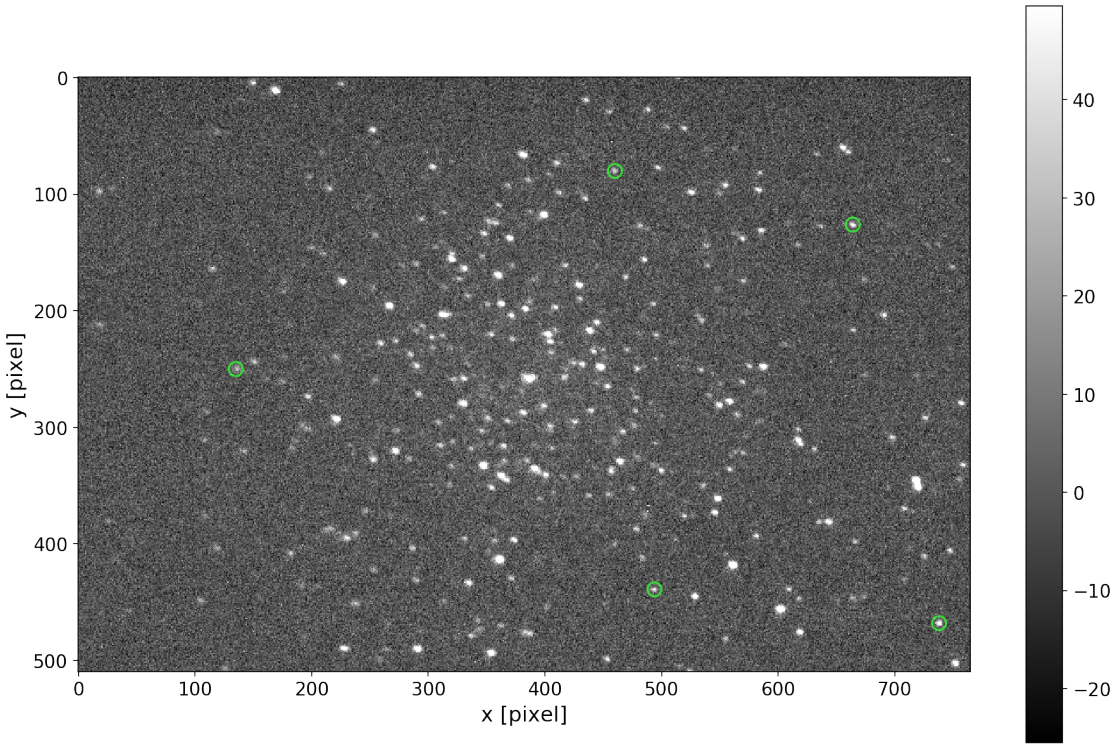
[2]:
```
def calculate_reference_stars_magnitudes(reference_stars, reference_image_data,
                                         photometric_dir, figure_number):
    """
    Calculates magnitued of the reference stars.

    Parameters
    ---------

    reference_stars: pandas.core.frame.DataFrame
        Positions of the reference stars

    reference_image_data: dict
        filter_name: str
            Name of the filter for the image
        aperture_radius: float
            Radius of the apertude to measure stars
        zero_point_mag: float
            The zero point magnitude term in the photometric equation.
        airmass_factor: float
            The air mass multiplier term in the photometric equation.

    figure_number: int
        The figure number that will be used in its caption.

    Returns
    --------
    list of float
        Magnitudes of the reference stars

    """

    # Set path to photometric image
    filter_name = reference_image_data["filter_name"]
    file_name = f"NGC_3201_{filter_name}_median_30.0s.fits"
    reference_image_path = os.path.join(photometric_dir, file_name)

    # Read photometric image
    reference_image = CCDData.read(reference_image_path)

    # Create apertures for reference stars
    # ----------

    reference_positions = list(
        zip(reference_stars["photometric_x"], reference_stars["photometric_y"])
```

```
    )

    aperture_radius = reference_image_data["aperture_radius"]
    reference_apertures = CircularAperture(reference_positions,␣
↪r=aperture_radius)

    # Show reference image with apertures
    title = f"Figrue {figure_number}: Reference stars in photometric image,␣
↪{filter_name} filter."
    show_image(image=reference_image, apertures=reference_apertures, title=title)
    plt.show()

    # Calculate fluxes of reference stars
    fluxes = aperture_photometry(reference_image, reference_apertures)

    # Calculate magnitudes of the stars using the photometric callibrations (Eq.␣
↪1-4)
    # -----------

    zero_point_mag = reference_image_data["zero_point_mag"]
    airmass_factor = reference_image_data["airmass_factor"]
    exposure_time = reference_image.header['EXPTIME']
    airmass = reference_image.header['AIRMASS']

    magnitudes = zero_point_mag - \
                    2.5 * np.log10(fluxes['aperture_sum'].value / exposure_time)␣
↪- \
                     airmass_factor * airmass

    # Round magnitudes
    magnitudes = [ round(magnitude, 3) for magnitude in magnitudes ]

    return magnitudes
```

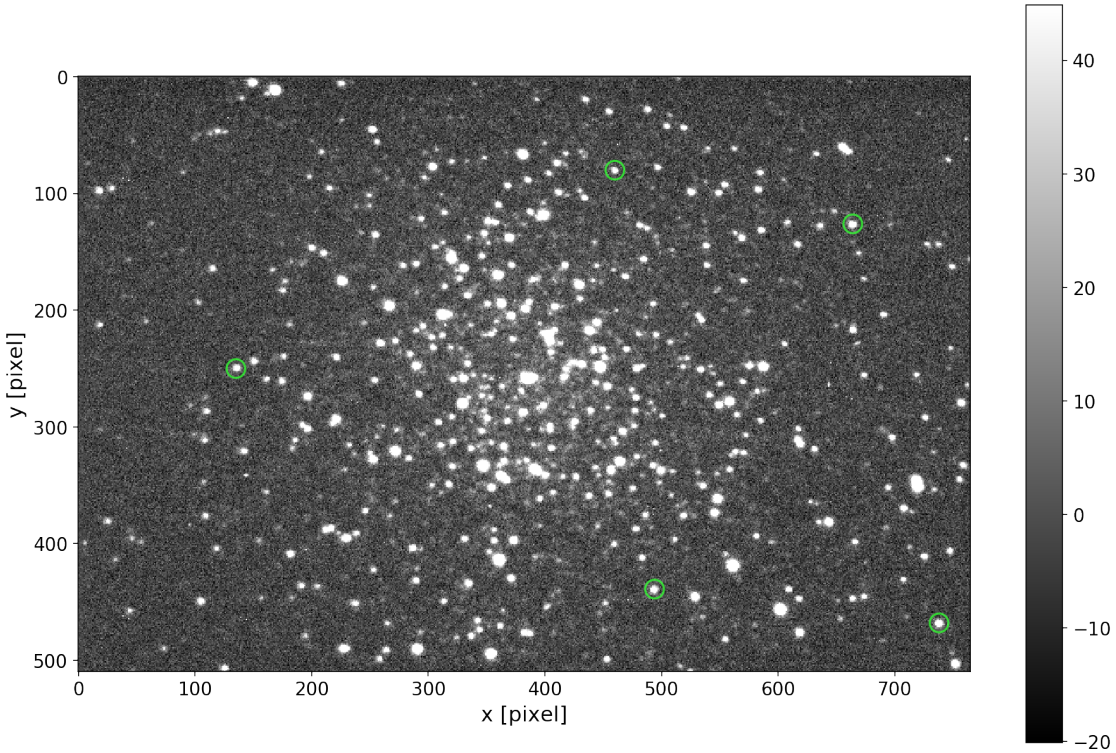
1.4 Calculate magnitudes of reference stars in photometric image

[3]:
```
# Load positions of reference stars
# ------

photometric_dir = "../050_scaling_and_combining/march_29_2018_stacked"
data_dir = "data"
reference_stars_path = os.path.join(data_dir, "reference_stars.csv")
reference_stars = pd.read_csv(reference_stars_path, index_col="star_number")
```

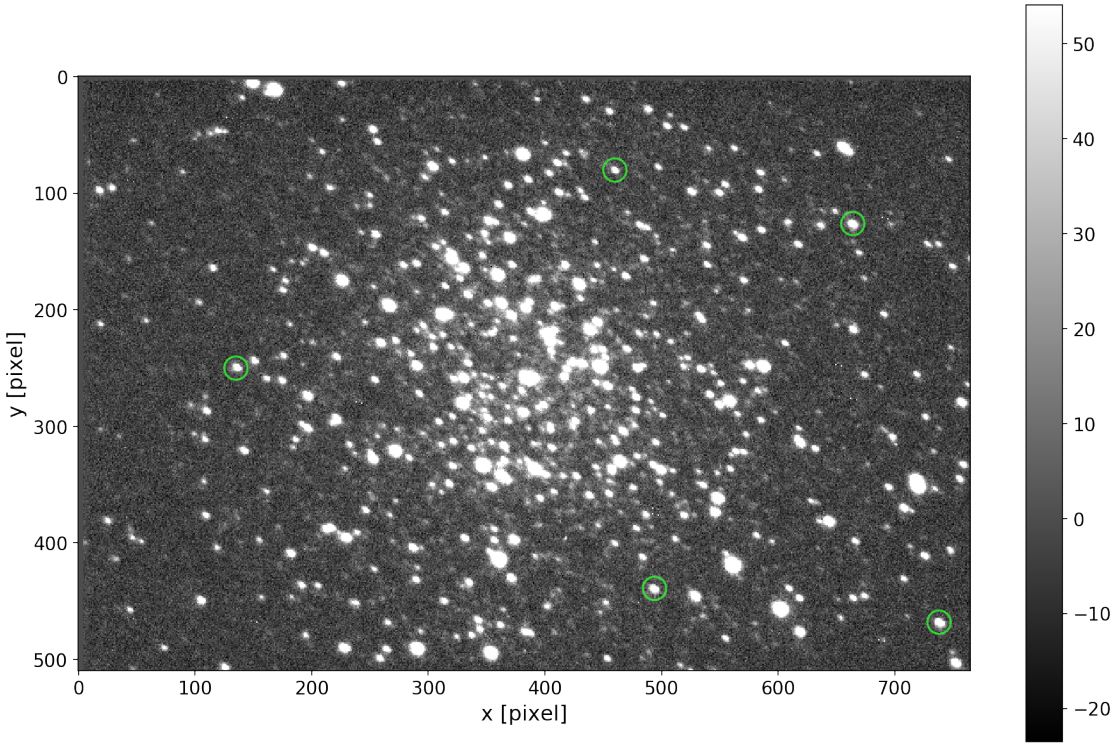
```
# Set properties of photometric images
reference_images = [
    dict(filter_name="B", aperture_radius=6, zero_point_mag=19.237,␣
 ↪airmass_factor=0.330),
    dict(filter_name="V", aperture_radius=8, zero_point_mag=19.696,␣
 ↪airmass_factor=0.210),
    dict(filter_name="R", aperture_radius=10, zero_point_mag=19.679,␣
 ↪airmass_factor=0.119),
    dict(filter_name="I", aperture_radius=10, zero_point_mag=19.079,␣
 ↪airmass_factor=0.134)
]

# Go over all photometric images
for i, reference_image_data in enumerate(reference_images):
    filter_name = reference_image_data["filter_name"]

    # Calculate magnitudes of the reference stars
    magnitudes = calculate_reference_stars_magnitudes(
        reference_stars=reference_stars,␣
 ↪reference_image_data=reference_image_data,
        photometric_dir=photometric_dir,
        figure_number=2+i)

    # Store magnitudes for the stars
    reference_stars[f"{filter_name.lower()}_mag"] = pd.Series(magnitudes,
                                                    index=reference_stars.index)


# Save magnitudes to a CSV file
reference_stars_path = os.path.join(data_dir, "reference_stars_magnitudes.csv")
reference_stars.to_csv(reference_stars_path)
```



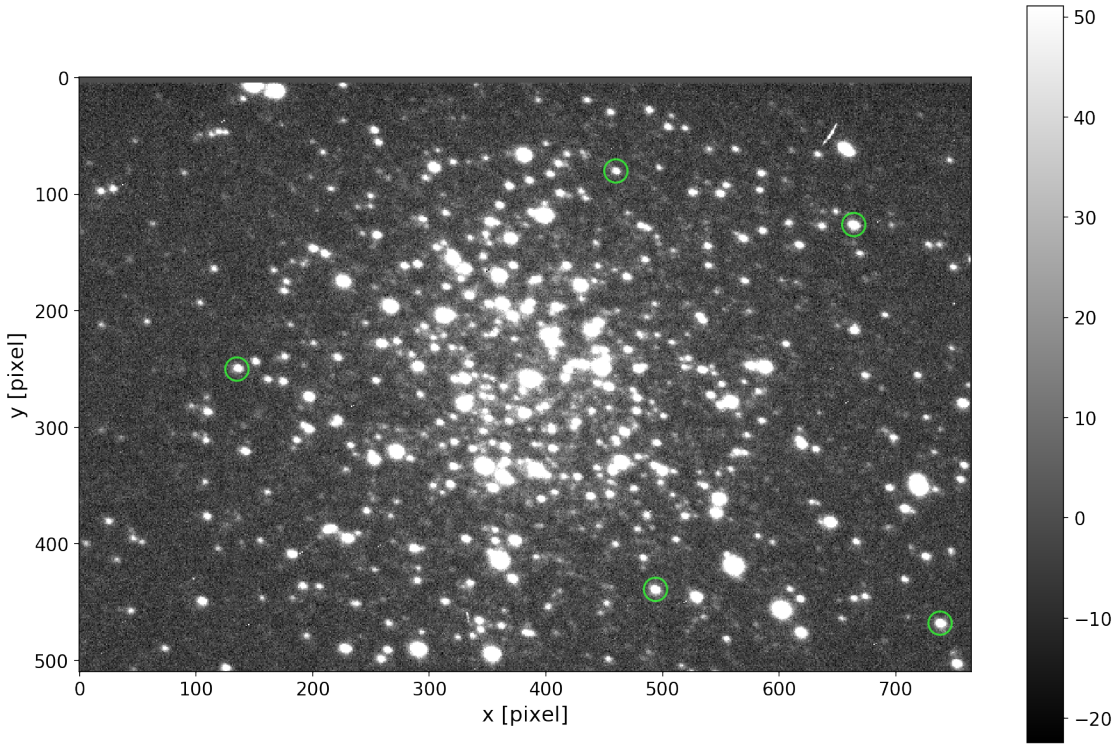
Figrue 2: Reference stars in photometric image, B filter.



Figrue 3: Reference stars in photometric image, V filter.



Figrue 4: Reference stars in photometric image, R filter.



Figrue 5: Reference stars in photometric image, I filter.

I look at Fig. 2-4 and check that green circles (apertures) of reference stars are located exactly like on Fig. 1. Also, I verify that the green circles have proper sizes such that they cover entire star with about 2 pixels of background around it.

1.5 Print magnitudes of reference stars in photometric image

I print calculated magnitudes of the reference stars to Fig. 2. I also show the magnitudes from SIMBAD for comparison.

[4]:
```
# Get only magnitude columns, since I don't want to print the entire table
df_printout = reference_stars[["b_mag_simbad", "b_mag", "v_mag_simbad",␣
 ↪"v_mag"]]

# Calculate difference between my magnitudes and SIMBAD's
# ------
delta_b = df_printout["b_mag_simbad"] - df_printout["b_mag"]
df_printout = df_printout.assign(delta_b=delta_b)
delta_v = df_printout["v_mag_simbad"] - df_printout["v_mag"]
df_printout = df_printout.assign(delta_v=delta_v)

# Set table column names
```

```
columns = ["#"] + list(df_printout.columns)

# Show the table
print("\nTable 2: Magnitudes of reference stars I calculated from photometric␣
 ↪images, as well as magnitudes from SIMBAD database.\n")
print(tabulate(df_printout, headers=columns, floatfmt=".2f", tablefmt="github"))
```

Table 2: Magnitudes of reference stars I calculated from photometric images, as well as magnitudes from SIMBAD database.

#	b_mag_simbad	b_mag	v_mag_simbad	v_mag	delta_b	delta_v
1	14.73	14.72	13.80	13.95	0.01	-0.15
2	15.15	15.14	14.02	14.17	0.01	-0.15
3	15.02	14.96	13.87	13.99	0.06	-0.12
4	15.56	15.49	14.77	14.84	0.07	-0.07
5	15.52	15.40	14.54	14.65	0.12	-0.11

1.6 Are my magnitudes any good?

The difference between our magnitudes and those from SIMBAD are:

[5]:
```
print(f"B mag: {df_printout['delta_b'].mean():.2f} ± {df_printout['delta_b'].
 ↪std():.2f}")
print(f"V mag: {df_printout['delta_v'].mean():.2f} ± {df_printout['delta_v'].
 ↪std():.2f}")
```

```
B mag: 0.06 ± 0.05
V mag: -0.12 ± 0.04
```

I don't know if those magnitude differences are too large or too small because I have not calculated measurement uncertainties, and SIMBAD did not have magnitude uncertainties either (for all stars except V magnitude of star 2).

1.7 Calculate magnitudes of stars in non-photometric night

Now that I calculated magnitued of reference stars in photometric night, I will use them to calculate magnitues of all stars in non-photometric night (March 9 2018).

Here is how it works.

1. Let m_2 be the magnitude of one reference star I calculated in photometric night, i.e. V=13.95 mag of the first star from Table 1.
2. I now use non-photometric night V image and measure the flux of the first reference star f_2.
3. Next, I measure flux f_1 of some other non-reference star I want to find magnitude for, in the same non-photometric image.
4. Then, I use the following equation to calculate the magnitude m_1 fo that star:

$$m_1 = m_2 - 2.5 \log(f_1/f_2). \quad (5)$$

5. Next, I repeat steps 1 through 4 for other four reference stars from Table 2. This will give me five measurements of m_1 for the same star.
6. Finally, I calculate the average and standard deviation of m_1, and that will be my final measured magnitude for the star.

[6]:
```
def get_distance(one, two):
    """
    Calculate distance in pixels between two points.

    Parameters
    ----------

    one, two: (x, y)
        Two points

    Returns
    -------
    float
        distance between `one` and `two`.
    """

    return np.sqrt((one[0] - two[0])**2 + (one[1] - two[1])**2)


def away_from_point(positions, distance, point):
    """
    Return positions that are `distance` pixels further from the `point`.

    Parameters
    ----------
    positions: list of (x, y)
        Positions.
    distance: float
        Min allowed distance from the center.
    point: (x, y)
        A coordinate.
```

```
    Returns
    -------
    list of (x, y)
        Positions that are `distance` pixels further away from the center
    """


    return [
        (x, y) for x, y in positions
        if get_distance(point, (x, y)) > distance
    ]


def away_from_center(positions, distance, image):
    """
    Return positions that are `distance` pixels firther from the center of the␣
 ↪image

    Parameters
    ----------
    positions: list of (x, y)
        Positions.
    distance: float
        Min allowed distance from the center.
    image: astropy.nddata.ccddata.CCDData
        Image.

    Returns
    -------
    list of (x, y)
        Positions that are `distance` pixels further away from the center
    """

    image_center = (image.data.shape[1] / 2, image.data.shape[0] / 2)

    return away_from_point(positions=positions, distance=distance,␣
 ↪point=image_center)


def away_from_each_other(positions, distance):
    """
    Exclude positions that are closer than `distance` to each other

    Parameters
    ----------
    positions: list of (x, y)
```

```
        Positions.
    distance: float
        Min allowed distance from the center.

    Returns
    -------
    list of (x, y)
        Positions that are further than `distance` pixels from each other.
    """

    return [
        (x, y) for x, y in positions
        if len(away_from_point(positions=positions, distance=distance,
                               point=(x, y))) == len(positions) - 1
    ]


def find_star_index(positions, distance, point):
    """
    Calculate the index of the object in the positions that is closer than␣
↪distance to `point`.

    Parameters
    ---------
    positions: list of (x, y)
        List of positions
    distance: float
        Distance.
    point: (x, y)
        A position.

    Returns
    -------
    int
        Index of the object in `positions`. Return None if none or more than one␣
↪objects found.
    """

    close_points = [
        i for i, position in enumerate(positions)
        if get_distance(point, position) < distance
    ]

    if len(close_points) == 1:
        return close_points[0]
    else:
        return None
```

```
def find_stars(image, aperture_radius, reference_stars, star_finder_threshold,
               min_distance_from_center, min_distance_between_stars):
    """
    Find positions of start in the `image`.

    Parameters
    ----------

    image: astropy.nddata.ccddata.CCDData
        An image to the stars in.

    aperture_radius: flost
        Radius of the star's aperture.

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

    star_finder_threshold: float
       The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    min_distance_from_center: float
       The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    min_distance_between_stars: float
        The minimum distance between the stars, in pixels.
        This is needed to avoid including stars that are too close together, and
        therefore have overlaping signal.

    Returns
    -------
    list of (x, y)
        Positions of stars in the image.
    """

    # Calculate mean, meadian and standard deviation of pixel values,
    # excluding extreme values that are less or more than 3 standard deviations␣
↪from
    # a venter value.
    mean, median, std = sigma_clipped_stats(image.data, sigma=3.0, maxiters=5)
```

```
    # Locate sources
    daofind = DAOStarFinder(fwhm=aperture_radius,
                            threshold=star_finder_threshold * std,
                            exclude_border=True)

    # Find the sourcesdistance
    sources = daofind(image.data - median)

    # Keep only stars away from crowded center of the cluster
    positions_all = [ (x, y) for x, y in zip(sources['xcentroid'],␣
↪sources['ycentroid']) ]

    # Remove stars in the center of the cluster, since it is crowded and
    # it is hard to distinguish stars
    positions_all = away_from_center(positions=positions_all,
                                     distance=min_distance_from_center, image=image)

    # Remove stars with overlaping apertures. This is needed because stars
    # that are close to each other will have overlapping fluxed that we can't␣
↪distinguish.
    positions = away_from_each_other(positions=positions_all,
                                       distance=min_distance_between_stars)

    # Find indices of the reference stars in the array of all stars
    reference_stars_map_all =␣
↪find_reference_stars(reference_stars=reference_stars,
                                                  positions=positions_all)

    reference_stars_map_filtered =␣
↪find_reference_stars(reference_stars=reference_stars,
                                                      positions=positions)

    # Check that all reference stars are still present
    for star_number, index in reference_stars_map_filtered.items():
        if index is None:
            # The reference star was filtered, add it back
            positions.append(positions_all[reference_stars_map_all[star_number]])


    return positions


def plot_stars(image, filter_name, positions, figure_number, aperture_radius,␣
↪reference_stars):
    """
    Plot the image and show positions of the stars on it.
```

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

    image: astropy.nddata.ccddata.CCDData
        An image to the stars in.

    filter_name: str
        Name of the liter, i.e. "V".

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

    figure_number: int
        The figure number shown in the plot's caption.

    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)
    title = f"Figrue {figure_number}: Stars in non-photometric image,␣
↪{filter_name} filter."
    show_image(image=image, apertures=apertures, title=title)

    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 * 1.7)
    reference_apertures.plot(color='#ff7777', lw=3, alpha=1)
    plt.show()


def find_reference_stars(reference_stars, positions):
    """
    Find indices of the reference stars in the array of all stars.

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

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

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

    Returns
    -------
    dict
        key: int, deference star number
        value: index of the star in array of all stars
    """

    reference_stars_to_all_stars_map = {}

    for index, reference_star in reference_stars.iterrows():
        position = reference_star[["non_photometric_x", "non_photometric_y"]].
↪values
        reference_stars_to_all_stars_map[index] =␣
↪find_star_index(positions=positions, distance=2, point=position)

        if index is  None:
            raise RuntimeError(f"Can not find reference star at position␣
↪{position} for filter {filter_name}")

    return reference_stars_to_all_stars_map


def measure_magnitudes_with_single_reference_star(
    image, filter_name, fluxes, reference_star,
    reference_star_index, reference_flux_index):
    """
    Measure magnitudes of all stars using single reference star.

    Parameters
    ----------
    image: astropy.nddata.ccddata.CCDData
        An image containing the stars.

    filter_name: str
        name of the filter

    fluxes: list of float
        List of fluxes for all stars.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of a single reference star.
```

```
    reference_star_index: int
        Reference star number

    reference_flux_index: int
        Index of the reference star in the list of all stars.

    Returns
    -------
    list of float
        List of magnitudes.
    """

    exposure_time = image.header['EXPTIME']
    m2 = reference_star[[f'{filter_name.lower()}_mag']].values[0]
    f2 = fluxes[reference_flux_index]

    magnitudes = [
        m2 - 2.5 * np.log10(flux / f2)
        for flux in fluxes
    ]

    # Verify that m1 is equal to m2 for the reference star
    # ----------

    m1 = magnitudes[reference_flux_index]

    if round(m1, 5) != round(m2, 5):
        raise RuntimeError(
            (
                f"Reference magnitudes are different: {m1}, {m2}, "
                f"reference star {reference_star_index}, {filter_name} filter"
            )
        )

    return magnitudes



def measure_magnitudes_with_all_reference_stars(image, filter_name,␣
 ↪reference_stars,
                                                positions, aperture_radius):
    """
    Measure magnitudes of all stars using all reference stars.

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

```
    image: astropy.nddata.ccddata.CCDData
        An image containing the stars.

    filter_name: str
        name of the filter

    fluxes: list of float
        List of fluxes for all stars.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of all reference stars.

    positions: list of (x, y)
        Positions of all stars.

    aperture_radius: float
        Aperture radius for measuring fluxes.

    Returns
    -------
    numpy.ndarray
        A 2D array containing fluxes of all stars for all reference stars.
        First index is the reference star, and second corresponds to all stars.
    """

    # Find indices of the reference stars in the array of all stars
    reference_stars_map = find_reference_stars(reference_stars=reference_stars,␣
↪positions=positions)

    # Measure flux
    apertures = CircularAperture(positions, r=aperture_radius)
    fluxes = aperture_photometry(image.data, apertures)['aperture_sum']

    # For each reference star, calculate the magnitudes of all stars,
    # storing results in a nested list
    magnitudes = [
        measure_magnitudes_with_single_reference_star(
            image=image, filter_name=filter_name,
            fluxes=fluxes, reference_star=reference_star,
            reference_star_index=index,
            reference_flux_index=reference_stars_map[index])

        for index, reference_star in reference_stars.iterrows()
    ]

    # Convert nested list to a 2D array
    return np.array(magnitudes)
```

```
def find_filter_magnitudes(filter_name, non_photometric_dir, reference_stars,
                           aperture_radius, figure_number,
                           star_finder_threshold, min_distance_from_center,
                           min_distance_between_stars):
    """
   Measure magnitudes of all stars in non-photometric image for the given filter.

    Parameters
    ---------

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

    non_photometric_dir: str
        Directory where non-photometric FITS files are located.

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions and magnitudes of all reference stars.

    aperture_radius: float
        Aperture radius for measuring fluxes.

    figure_number: int
        The figure number that will be shown in plot's caption.

    star_finder_threshold: float
       The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    min_distance_from_center: float
       The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    min_distance_between_stars: float
        The minimum distance between the stars, in pixels.
        This is needed to avoid including stars that are too close together, and
        therefore have overlaping signal.

    Returns
    ------
    numpy.ndarray
```

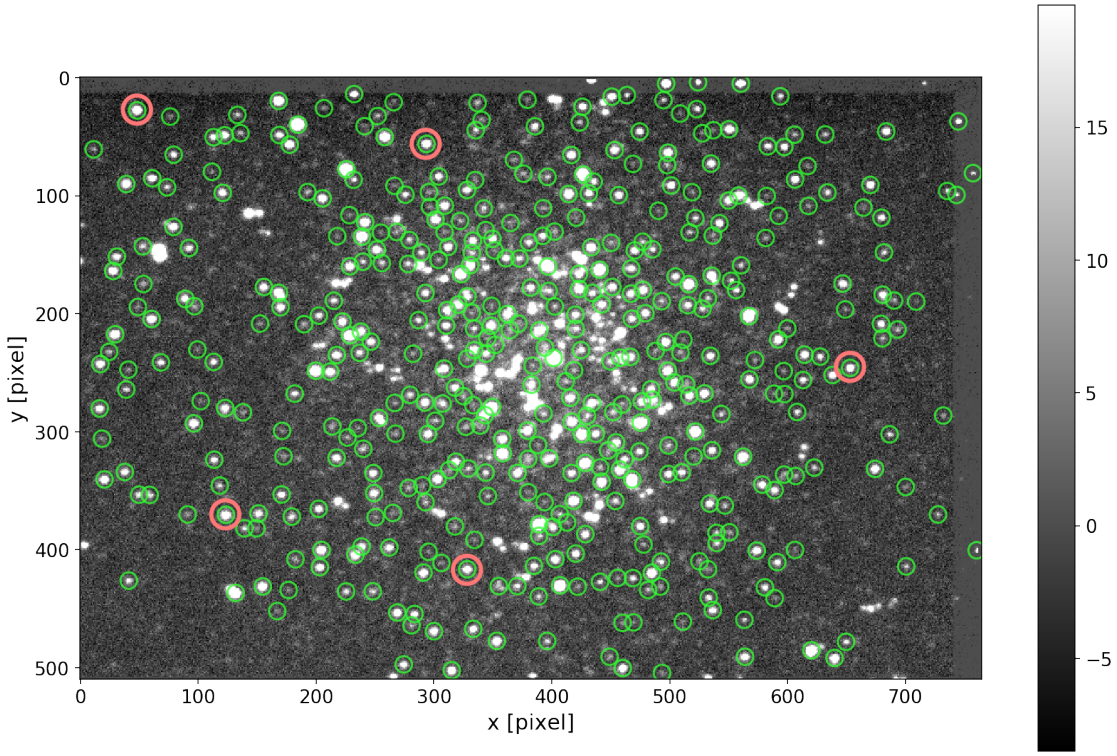
```
        A 2D array containing positions of all stars, their mean magnitudes and␣
↪standard deviations.

        The first index is:
            0: position (x, y)
            1: magnitude float
            2: magnitude standard deviation

        The second index the the index of the star.
    """

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

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

    # Find positions of stars
    positions = find_stars(image=image, aperture_radius=aperture_radius,
                           reference_stars=reference_stars,
                           star_finder_threshold=star_finder_threshold,
                           min_distance_from_center=min_distance_from_center,
                          min_distance_between_stars=min_distance_between_stars)

    # Plot the apertures
    plot_stars(image=image, filter_name=filter_name,
               positions=positions, figure_number=figure_number,
               aperture_radius=aperture_radius,
               reference_stars=reference_stars)

    all_magnitudes = measure_magnitudes_with_all_reference_stars(
        image=image, filter_name=filter_name, reference_stars=reference_stars,
        positions=positions, aperture_radius=aperture_radius)

    return np.array(
        [positions, all_magnitudes.mean(axis=0), all_magnitudes.std(axis=0)])
```

1.8 Measure magnitudes of all stars for each filter

[7]:
```
def find_magnitudes(filter_names, non_photometric_dir,
                    reference_stars, aperture_radius,
                    star_finder_threshold, figure_number,
                    min_distance_from_center,
                    min_distance_between_stars):
    """
```

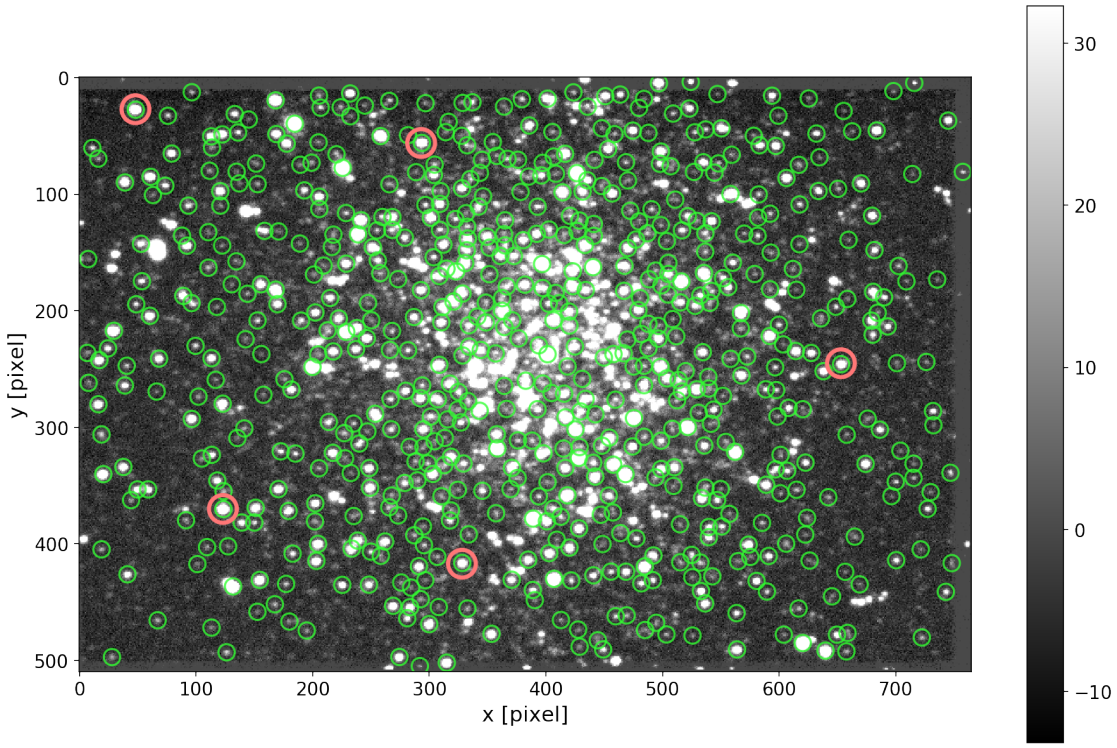
```
    Measure magnitudes of all stars for all filters.

    Parameters
    ----------

    filter_names: list of str
        Names of the image filters, i.e. ["V", "R"].

    non_photometric_dir: str
        Path to directory where the non-photometric FITS files are located.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of all reference stars.

    aperture_radius: float
        Radius of the aperture used to measure fluxes of the stars.

    star_finder_threshold: float
       The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    figure_number: int
        The number used in plot's caption.

    min_distance_from_center: float
       The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    Returns
    -------
    dict
        key: filter name name, i.e. "V"
        value: numpy.ndarray
            A 2D array containing positions of all stars, their mean magnitudes␣
↪and standard deviations.

            The first index is:
                0: position (x, y)
                1: magnitude float
                2: magnitude standard deviation

            The second index the the index of the star.
    """
```

```
    return {
        filter_name: find_filter_magnitudes(
                        filter_name=filter_name,
                        non_photometric_dir=non_photometric_dir,
                        reference_stars=reference_stars,
                        aperture_radius=aperture_radius,
                        figure_number=figure_number + i,
                        star_finder_threshold=star_finder_threshold,
                        min_distance_from_center=min_distance_from_center,
                        min_distance_between_stars=min_distance_between_stars)

        for i, filter_name in enumerate(filter_names)
    }


non_photometric_dir = "../050_scaling_and_combining/march_09_2018_stacked"
filter_names = ["B", "V", "R", "I"]

filter_magnitudes = find_magnitudes(filter_names=filter_names,
                                    non_photometric_dir=non_photometric_dir,
                                    reference_stars=reference_stars,
                                    aperture_radius=7,
                                    star_finder_threshold=5,
                                    figure_number=6,
                                    min_distance_from_center=1,
                                    min_distance_between_stars=8.4)

print("Finished measuring magnitudes")
```



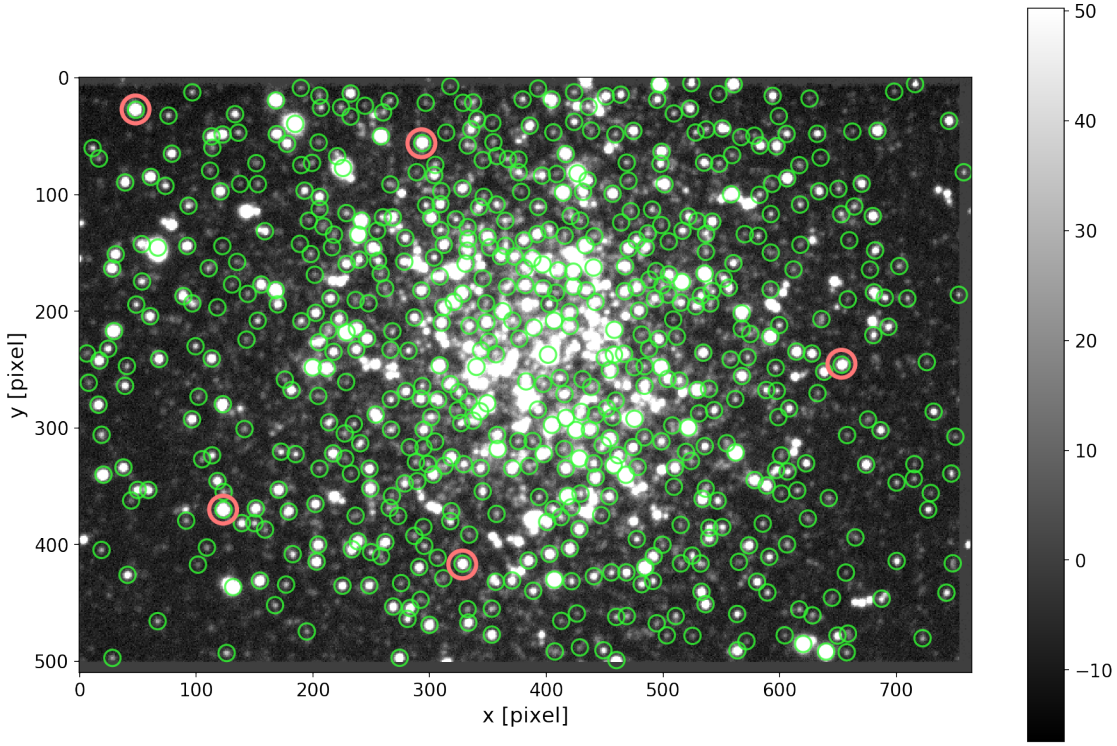
Figrue 6: Stars in non-photometric image, B filter.

```
/opt/miniconda3/envs/obsastro2020/lib/python3.7/site-
packages/ipykernel_launcher.py:315: RuntimeWarning: invalid value encountered in
log10
```



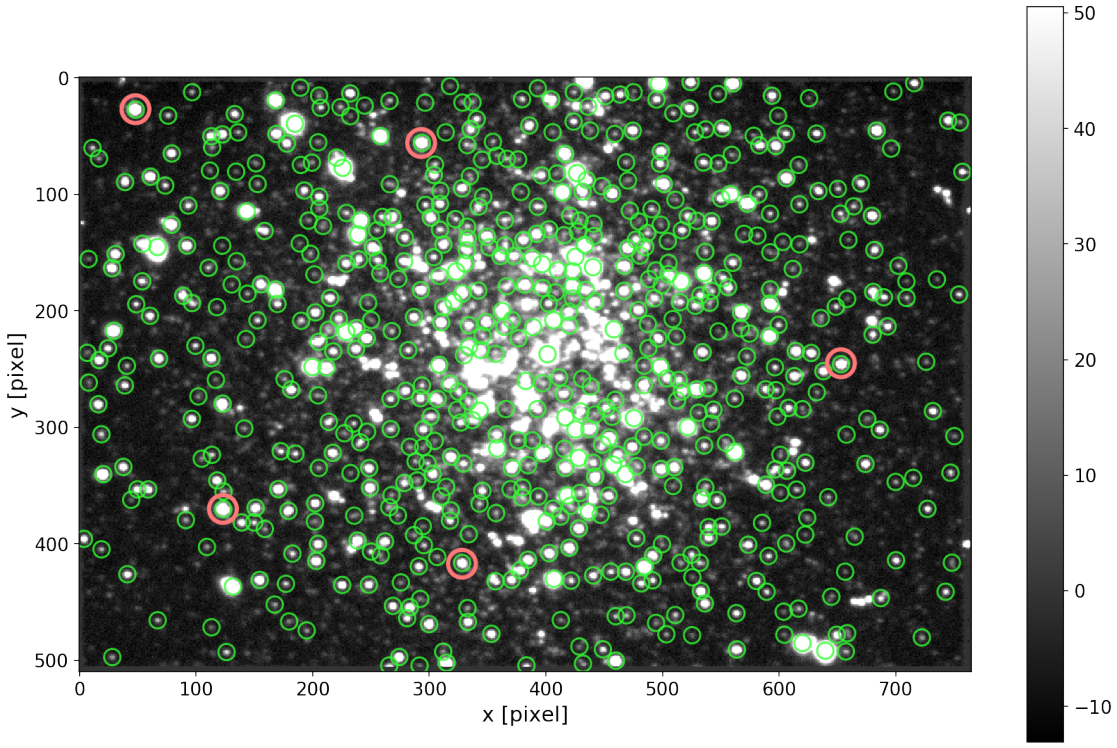
Figrue 7: Stars in non-photometric image, V filter.

```
/opt/miniconda3/envs/obsastro2020/lib/python3.7/site-
packages/ipykernel_launcher.py:315: RuntimeWarning: invalid value encountered in
log10
```



Figrue 8: Stars in non-photometric image, R filter.

```
/opt/miniconda3/envs/obsastro2020/lib/python3.7/site-
packages/ipykernel_launcher.py:315: RuntimeWarning: invalid value encountered in
log10
```



Figrue 9: Stars in non-photometric image, I filter.

```
Finished measuring magnitudes
```

```
/opt/miniconda3/envs/obsastro2020/lib/python3.7/site-
packages/ipykernel_launcher.py:315: RuntimeWarning: invalid value encountered in
log10
```

1.9 Are apertures any good?

I look at Fig. 6-9 and check these things:

1. Are the apertures (small green circles) match the stars well? Answer: yes
2. Are the apertures cover the stars completely? Answer: for most stars - yes. There are several fat oversaturated stars stars, especially in filter R and I (Fig. 8 and 9). For those stars, the size of the aperture could be larger. It would be nice to have a way to programmatically detect those stars and make apertures larger for those.
3. Are the reference stars (larger red circles) match the green circles? Answer: yes.
4. There there multiple stars covered by the same aperture? Answer: no, for majority of stars.

Conclusion: I'm happy with star detection and aperture choice.

1.10 Verify standard deviations of magnitudes are the same for each filter

We measured magnitude of each star five times, using five reference stars. The standard devistion of these five numbers should be identical for all stars. Why? Ok, let's look at Eq. 5 again:

$m_1 = m_2 - 2.5\log(f_1/f_2) = m_2 + 2.5\log(f_2) - 2.5\log(f_1)$.

The measured flux of the star f_1 is the same between five measurements. So its five magnitudes m_1 will be equal to $m_2 + 2.5\log(f_2)$ plus constant value $-2.5\log(f_1)$. Therefore, the variation between five measurements is only due to variation of the $m_2 + 2.5\log(f_2)$ values. These values come from reference stars, which are the same for all stars. Thefore, all stars should have the same variation between five measurements. Or, in other words, the standard deviation of these five measurements will be the same.

Let's verify this in code.

[8]:
```
# Loop over all filters
for filter_name, magnitudes in filter_magnitudes.items():
    # Raise error if standard deviations of magnitudes are different for each␣
↪filter
    if magnitudes[2].std() > 1e-10:
        raise RuntimeError(f"ERROR: standard deviations vary for {filter_name}␣
↪filter")
```

1.11 Print standard deviations of five magnitudes

Now that we checked that standard deviations of five magnitudes for each filter are the same for all stars, let's print them.

[9]:
```
def print_standard_deviations(filter_magnitudes):
    """
    Print standard deviations of five measured magnitudes for each filter.

    Parameters
    ----------

    filter_magnitudes: dict
        key: filter name name, i.e. "V"
        value: numpy.ndarray
            A 2D array containing positions of all stars, their mean magnitudes␣
↪and standard deviations.

            The first index is:
                0: position (x, y)
                1: magnitude float
                2: magnitude standard deviation

            The second index the the index of the star.
```

```
    """

    print("\nTable 3: Standard deviation of five magnitudes")

    # Loop over all filters
    for filter_name, magnitudes in filter_magnitudes.items():
        # I only need standard deviatino for one star
        # since they are identical for other stars, as I checked previously
        std = magnitudes[2][0]

        print(f'{filter_name}: {std:.3f}')

        if std > 0.2:
            raise RuntimeError(f"ERROR: magnitude standard deviation {std:.4f}␣
↪mag is too large for {filter_name} filter")


print_standard_deviations(filter_magnitudes=filter_magnitudes)
```

```
Table 3: Standard deviation of five magnitudes
B: 0.086
V: 0.037
R: 0.036
I: 0.041
```

The standard deviations are 0.1 mag or smaller (Table 3), I'm happy with the results. The standard deviations is largest of B filter, probably because we used only one photometric image for it. Hmm, maybe I should not have removed that other blurry B image. I regret I did that.

> Maybe all one can do is hope to end up with the right regrets.
>
> Arthur Miller, The Ride Down Mt. Morgan

1.12 Save the magnitudes to a CSV file

The next step is to save the magnitudes of the stars in a CSV file. Here I will store the magnitudes in a table (Pandas data frame). Each row is a different star. The columns are, x-y positions, and B, V, R, I magnitudes.

The only nuiance here is that x-y positions of the same star will vary slightly between different filters, say (123, 42) for V image and (124, 41) for R image. But even though positions are diffrent, it's still the same star. So we want a single row in the table, not two rows. To achive that I compare coordinates of stars between filters, and consider them to correspond to the same star if the distance between the coordinates is smaller than two pixels.

[10]:
```
def save_magnitudes(filter_magnitudes, file_path):
    """
    Store positions and magnitudes of stars for all filters to a CSV file.
```

```
    Parameters
    ---------
    filter_magnitudes: dict
        key: filter name name, i.e. "V"
        value: numpy.ndarray
            A 2D array containing positions of all stars, their mean magnitudes␣
↪and standard deviations.

            The first index is:
                0: position (x, y)
                1: magnitude float
                2: magnitude standard deviation

            The second index the the index of the star.

    file_path: str
        Path to the CSV file where the magnitudes are saved.

    Returns
    -------
    pandas.core.frame.DataFrame
        A table containing magnitudes of all stars.
    """

    filter_names = filter_magnitudes.keys()

    # Set column name that include positions and magnitudes for all filters
    filter_columns = [ f'{name.lower()}_mag' for name in filter_names ]
    column_names = ["x", "y"] + filter_columns

    # Create empty table
    df = pd.DataFrame(columns=column_names)

    # Maximum distance between stars for different filters
    max_star_distance = 2

    # For each filter, store magnitudes of stars in the table
    for filter_name in filter_names:
        # Get the list of magnitudes for the current filter
        magnitudes = filter_magnitudes[filter_name]

        # Loop over all positions of the stars for the current filter
        for i, position in enumerate(magnitudes[0, :]):
            # Get star's magnitude and its standard deviation
            mag = magnitudes[1, i]
            mag_std = magnitudes[2, i]
```

```
        # Check if the star is already present in the table
        # because its magnitude was measured for other filters
        # --------

        positions = df[["x", "y"]].values.tolist()

        star_index = find_star_index(positions=positions,
                                     distance=max_star_distance,
                                     point=position)

        if star_index is None:
            # This is a new star, add a row to the table
            # ------

            row = {
                'x': int(round(position[0])),
                'y': int(round(position[1])),
                f'{filter_name.lower()}_mag': round(mag, 2),
            }

            df = df.append(row, ignore_index=True)
        else: # This star has already been added to the table
            # Find star's row in the table
            row = df.iloc[star_index]

            # Get stars's x-y position from the table
            # ------

            existing_xy = row[["x", "y"]].values.tolist()
            existing_x = existing_xy[0]
            existing_y = existing_xy[1]

            # Verify that star's position in the table is similar to
            # star's position for current filter
            if abs(existing_x - position[0]) > max_star_distance or \
                abs(existing_y - position[1]) > max_star_distance:

                raise RuntimeError(
                    (
                        f"Non matching positions in magnitudes table: "
                        f"{existing_xy} and {position}"
                    )
                )

            # Update existing table row with star's magnitude
            row[f'{filter_name.lower()}_mag'] = round(mag, 2)
```

```
    # Save coordinates as integers, not floats
    df.x = df.x.astype(int)
    df.y = df.y.astype(int)

    # Save magnitudes to a CSV file
    df.to_csv(file_path, index=False)

    return df


# Save magnitudes to a CSV file
magnitudes_path = os.path.join(data_dir, "magnitudes.csv")
df_magnitudes = save_magnitudes(filter_magnitudes=filter_magnitudes,␣
 ↪file_path=magnitudes_path)

print(f"Magnitudes are saved to: {magnitudes_path}")
```

```
Magnitudes are saved to: data/magnitudes.csv
```

1.13 Print magnitudes I calculated

The magnitudes of the stars I calculated are shown in Table 4.

[11]:
```
# Print the table of magnitude differences
print("\nTable 4: Magnitudes of stars we measured.\n")

print(tabulate(df_magnitudes,
               headers=["#"] + list(df_magnitudes),
               floatfmt=".2f", tablefmt="github"))
```

```
Table 4: Magnitudes of stars we measured.
```

#	x	y	b_mag	v_mag	r_mag	i_mag
0	524.00	4.00	16.66	16.42	15.17	13.74
1	497.00	5.00	14.23	13.96	12.39	11.08
2	560.00	5.00	14.78	nan	13.71	12.69
3	232.00	14.00	15.85	15.09	14.63	14.24
4	464.00	15.00	19.50	16.31	15.56	14.87
5	451.00	16.00	15.35	14.90	14.59	14.22
6	594.00	16.00	18.14	16.10	15.34	14.64
7	379.00	19.00	nan	17.74	16.77	16.02
8	495.00	19.00	nan	nan	15.49	14.65
9	168.00	20.00	14.51	13.59	12.98	12.27
10	266.00	21.00	nan	19.71	17.66	16.84

11	337.00	21.00	18.80	16.93	16.45	16.22
12	426.00	25.00	15.83	nan	nan	nan
13	207.00	26.00	nan	18.24	16.96	16.12
14	523.00	27.00	17.40	15.76	15.07	14.35
15	48.00	28.00	14.84	13.98	13.43	12.80
16	509.00	30.00	nan	nan	18.74	16.93
17	133.00	32.00	22.46	16.64	15.98	15.20
18	252.00	32.00	19.18	nan	nan	nan
19	76.00	33.00	nan	nan	19.55	18.05
20	340.00	36.00	19.02	nan	15.76	15.14
21	745.00	37.00	16.57	15.52	14.99	14.46
22	424.00	38.00	18.61	nan	15.84	15.16
23	185.00	40.00	13.19	11.93	11.12	10.21
24	241.00	41.00	nan	nan	nan	nan
25	386.00	41.00	16.27	15.36	14.90	14.39
26	335.00	44.00	16.63	nan	14.98	14.33
27	550.00	44.00	15.31	14.91	14.61	14.23
28	475.00	46.00	16.33	15.25	14.59	13.85
29	537.00	44.00	nan	20.44	17.64	16.75
30	684.00	46.00	15.78	14.65	14.01	13.26
31	136.00	47.00	nan	17.84	18.73	nan
32	528.00	47.00	nan	18.99	17.51	16.65
33	606.00	48.00	19.51	nan	15.98	15.40
34	632.00	48.00	nan	16.84	16.10	15.32
35	123.00	49.00	15.51	14.90	14.47	14.03
36	169.00	49.00	16.03	15.00	14.38	13.63
37	258.00	50.00	14.44	13.48	12.83	12.09
38	113.00	50.00	15.99	15.18	14.68	14.15
39	294.00	56.00	15.20	14.19	13.55	12.83
40	178.00	57.00	15.23	14.88	14.68	14.31
41	583.00	58.00	16.80	15.66	15.05	14.39
42	597.00	59.00	16.22	15.15	14.50	13.78
43	11.00	61.00	nan	20.44	19.06	18.56
44	453.00	61.00	15.41	14.84	nan	nan
45	499.00	64.00	15.00	14.56	14.16	13.78
46	79.00	65.00	16.84	15.60	15.04	14.35
47	417.00	65.00	15.26	14.23	13.56	12.77
48	368.00	70.00	17.84	16.95	16.43	15.75
49	535.00	73.00	15.74	15.45	15.38	15.19
50	469.00	73.00	nan	18.78	17.78	16.66
51	498.00	74.00	17.06	16.12	15.39	14.71
52	617.00	75.00	nan	19.02	18.36	18.13
53	226.00	78.00	13.46	12.00	11.07	10.07
54	112.00	80.00	nan	nan	nan	nan
55	757.00	81.00	17.79	17.81	17.55	15.52
56	376.00	81.00	17.04	nan	15.49	nan
57	427.00	82.00	13.45	12.13	11.26	10.34
58	304.00	84.00	15.79	15.09	14.65	14.07

59	396.00	84.00	16.68	15.70	15.10	14.34
60	61.00	85.00	15.67	14.78	14.30	13.69
61	606.00	86.00	15.59	14.51	13.84	13.07
62	232.00	86.00	16.43	nan	nan	nan
63	335.00	87.00	17.54	16.57	nan	nan
64	436.00	88.00	16.02	14.96	13.94	12.94
65	39.00	90.00	15.19	14.95	14.87	14.65
66	501.00	91.00	15.50	nan	13.76	12.97
67	670.00	91.00	15.75	15.48	15.35	15.15
68	266.00	92.00	nan	17.66	16.76	16.04
69	74.00	93.00	17.42	16.67	16.56	16.40
70	328.00	95.00	15.74	15.14	14.68	14.17
71	736.00	96.00	18.95	nan	nan	nan
72	193.00	97.00	17.85	16.49	15.79	15.05
73	296.00	97.00	17.43	nan	nan	nan
74	519.00	97.00	nan	nan	nan	nan
75	634.00	97.00	17.22	16.15	15.49	14.86
76	121.00	98.00	15.71	15.03	14.60	14.11
77	431.00	98.00	15.56	14.79	14.20	13.50
78	276.00	99.00	16.61	nan	nan	nan
79	414.00	99.00	14.61	13.56	12.87	12.07
80	457.00	99.00	15.59	14.66	14.01	13.26
81	743.00	99.00	17.47	nan	nan	nan
82	558.00	100.00	14.65	13.78	13.16	12.43
83	582.00	100.00	19.33	17.00	15.89	15.14
84	206.00	102.00	15.27	15.03	14.88	14.64
85	550.00	104.00	15.18	nan	nan	13.67
86	309.00	108.00	15.23	14.91	14.74	14.43
87	297.00	109.00	17.68	16.78	16.07	15.44
88	618.00	109.00	21.20	17.62	17.12	16.65
89	342.00	111.00	16.47	15.65	15.17	14.58
90	664.00	110.00	nan	nan	16.81	16.36
91	393.00	111.00	17.13	16.36	15.72	14.96
92	490.00	113.00	nan	17.48	16.61	15.81
93	228.00	116.00	19.56	17.34	16.47	15.40
94	592.00	117.00	nan	19.85	17.89	16.77
95	522.00	119.00	17.00	15.87	15.42	14.78
96	680.00	119.00	16.33	15.28	14.64	13.89
97	301.00	120.00	14.88	14.46	14.17	13.72
98	421.00	119.00	17.10	16.16	15.66	14.98
99	322.00	121.00	17.02	16.46	16.02	15.43
100	242.00	122.00	14.67	13.67	12.99	12.21
101	365.00	123.00	17.01	16.07	15.58	14.89
102	542.00	123.00	15.74	15.09	14.64	14.09
103	79.00	126.00	15.31	nan	nan	13.05
104	344.00	129.00	16.95	nan	nan	nan
105	309.00	131.00	17.17	16.49	15.83	15.11
106	402.00	130.00	16.92	15.94	15.37	14.67

107	255.00	131.00	nan	18.19	17.30	16.15
108	517.00	131.00	17.85	16.38	15.74	15.01
109	268.00	131.00	16.52	15.83	nan	nan
110	218.00	134.00	20.53	17.97	17.79	nan
111	392.00	134.00	15.89	14.89	14.26	13.49
112	537.00	134.00	17.40	16.20	15.57	nan
113	239.00	135.00	14.10	12.99	12.26	11.44
114	581.00	136.00	nan	18.06	17.80	nan
115	350.00	137.00	15.52	14.64	14.03	13.32
116	279.00	138.00	16.79	15.94	15.37	14.68
117	333.00	138.00	15.48	14.66	14.05	13.36
118	477.00	139.00	16.85	15.91	15.18	14.42
119	381.00	140.00	15.93	15.15	14.61	13.94
120	450.00	140.00	16.59	nan	nan	nan
121	53.00	143.00	16.54	15.66	14.71	13.55
122	312.00	143.00	15.43	14.87	14.42	13.89
123	92.00	144.00	16.28	15.23	14.57	13.82
124	433.00	144.00	14.81	13.96	13.33	12.59
125	252.00	146.00	15.22	14.61	14.16	13.64
126	485.00	145.00	16.26	nan	14.73	14.02
127	351.00	146.00	16.94	16.10	15.33	14.68
128	470.00	146.00	15.69	14.74	14.11	13.36
129	289.00	148.00	16.22	nan	nan	nan
130	682.00	148.00	17.25	16.19	15.61	14.97
131	332.00	148.00	15.98	15.23	14.59	13.96
132	31.00	152.00	16.01	nan	14.57	13.90
133	361.00	153.00	15.59	14.72	14.14	13.41
134	372.00	153.00	15.97	15.17	14.59	13.90
135	304.00	154.00	17.59	16.60	15.95	15.15
136	240.00	156.00	16.61	nan	15.22	14.52
137	256.00	157.00	16.78	16.03	15.59	14.98
138	278.00	158.00	16.08	nan	14.60	13.76
139	331.00	159.00	14.74	14.03	13.40	12.73
140	561.00	159.00	17.23	16.11	15.48	14.74
141	229.00	160.00	15.19	14.91	14.75	14.38
142	397.00	160.00	13.66	12.66	11.97	11.16
143	467.00	162.00	15.56	14.65	14.04	13.31
144	441.00	163.00	13.55	12.24	11.39	10.44
145	28.00	164.00	15.36	nan	14.43	13.95
146	323.00	166.00	13.96	12.84	nan	11.26
147	423.00	166.00	14.94	14.13	13.50	12.77
148	505.00	168.00	15.80	15.10	14.51	13.76
149	536.00	168.00	14.67	13.64	12.96	12.17
150	552.00	172.00	17.08	16.02	15.39	14.77
151	647.00	174.00	15.64	nan	nan	nan
152	53.00	175.00	17.66	16.80	16.18	15.26
153	516.00	175.00	14.04	12.87	12.09	11.22
154	156.00	177.00	15.65	14.97	14.50	13.88

155	451.00	177.00	15.31	nan	nan	nan
156	382.00	178.00	15.43	14.58	13.96	13.25
157	423.00	179.00	14.69	13.99	13.48	12.84
158	477.00	180.00	15.07	nan	14.46	nan
159	556.00	180.00	16.47	15.47	14.82	14.10
160	397.00	181.00	15.78	15.09	14.55	13.86
161	168.00	182.00	14.63	13.80	13.23	12.52
162	293.00	182.00	15.79	14.95	14.35	13.64
163	434.00	182.00	15.78	15.03	14.45	13.73
164	467.00	183.00	15.37	14.70	14.22	13.61
165	681.00	184.00	15.62	14.97	14.52	13.95
166	328.00	185.00	15.45	14.62	14.01	13.33
167	89.00	187.00	15.68	15.16	14.72	14.18
168	532.00	187.00	16.76	15.88	15.29	14.59
169	215.00	189.00	16.40	15.67	15.11	14.38
170	493.00	189.00	16.43	nan	15.02	14.39
171	690.00	189.00	18.37	16.68	16.02	15.28
172	709.00	190.00	nan	nan	17.98	17.26
173	321.00	192.00	14.95	14.07	13.47	12.75
174	442.00	192.00	15.15	nan	14.16	13.57
175	349.00	193.00	17.17	16.47	15.87	nan
176	515.00	193.00	15.86	15.43	15.17	14.69
177	49.00	194.00	19.54	17.84	17.07	16.33
178	97.00	194.00	16.90	16.16	15.68	15.08
179	403.00	194.00	16.75	16.06	15.45	14.68
180	170.00	195.00	15.46	15.31	15.07	14.82
181	528.00	196.00	16.48	15.74	15.29	14.73
182	649.00	196.00	nan	18.43	nan	16.53
183	311.00	197.00	15.17	14.79	14.43	13.87
184	332.00	199.00	16.46	nan	nan	nan
185	362.00	200.00	14.77	14.05	13.60	13.00
186	479.00	200.00	15.37	14.83	14.38	13.81
187	420.00	201.00	15.40	14.81	14.34	13.74
188	202.00	202.00	16.42	15.45	14.86	14.17
189	567.00	202.00	13.87	13.61	13.43	13.11
190	60.00	204.00	15.27	15.13	15.08	14.95
191	468.00	204.00	15.62	nan	nan	nan
192	287.00	205.00	16.38	15.48	14.93	14.18
193	222.00	207.00	15.33	15.15	14.90	14.47
194	152.00	208.00	20.36	17.63	16.98	16.16
195	372.00	208.00	16.18	15.48	14.93	14.29
196	679.00	209.00	15.96	14.94	nan	nan
197	190.00	209.00	17.62	16.55	16.19	15.66
198	310.00	210.00	15.62	14.72	14.10	13.38
199	348.00	210.00	14.92	14.49	14.19	13.75
200	334.00	213.00	16.14	15.34	14.75	14.09
201	600.00	212.00	20.08	17.46	16.45	15.71
202	420.00	213.00	15.70	15.00	14.43	13.77

203	364.00	214.00	16.03	15.29	nan	14.10
204	389.00	214.00	14.50	nan	13.65	13.14
205	693.00	213.00	17.18	16.40	15.74	15.01
206	238.00	215.00	15.04	14.40	13.77	13.12
207	30.00	217.00	14.90	13.98	13.37	12.64
208	229.00	218.00	14.04	12.98	12.27	11.43
209	345.00	220.00	16.47	nan	nan	14.62
210	680.00	221.00	17.68	16.57	15.91	15.14
211	512.00	222.00	18.83	17.21	16.62	16.04
212	592.00	222.00	14.99	13.97	13.30	12.54
213	246.00	224.00	15.59	14.79	14.14	13.41
214	497.00	224.00	15.10	14.73	14.46	14.07
215	352.00	226.00	16.28	nan	15.05	nan
216	394.00	229.00	15.47	14.60	nan	nan
217	99.00	230.00	nan	18.58	17.70	16.70
218	334.00	231.00	14.77	13.91	nan	12.60
219	425.00	231.00	15.65	14.88	nan	nan
220	488.00	231.00	16.95	16.57	16.12	15.44
221	24.00	232.00	17.78	16.73	16.14	15.41
222	237.00	233.00	16.13	15.40	14.78	14.05
223	509.00	233.00	16.99	nan	nan	nan
224	282.00	234.00	17.43	16.48	15.97	15.20
225	344.00	234.00	15.58	14.81	14.16	13.43
226	218.00	235.00	15.32	15.13	14.99	14.68
227	614.00	235.00	15.59	14.86	14.31	13.69
228	535.00	236.00	15.72	15.03	14.54	13.96
229	628.00	236.00	16.67	15.59	14.92	14.17
230	402.00	237.00	13.21	12.08	11.35	10.48
231	458.00	237.00	14.49	13.55	12.91	nan
232	467.00	237.00	14.97	14.29	13.76	13.08
233	328.00	238.00	16.23	15.59	nan	13.66
234	450.00	240.00	15.47	14.62	13.88	nan
235	572.00	239.00	18.27	17.06	16.54	16.03
236	68.00	241.00	16.50	15.47	14.89	14.20
237	113.00	241.00	16.44	15.49	14.85	14.10
238	17.00	243.00	15.51	15.06	14.75	14.39
239	384.00	244.00	16.26	nan	nan	nan
240	309.00	246.00	14.91	13.97	13.35	12.60
241	653.00	246.00	15.40	14.58	14.03	13.35
242	41.00	247.00	nan	20.17	19.53	23.46
243	200.00	248.00	13.99	12.80	12.05	11.18
244	419.00	248.00	16.39	15.83	nan	nan
245	498.00	248.00	14.74	13.79	13.17	12.42
246	597.00	249.00	nan	18.48	17.10	16.24
247	212.00	249.00	15.10	nan	13.57	12.81
248	638.00	252.00	15.63	15.02	14.55	14.05
249	568.00	256.00	15.49	14.81	14.34	13.77
250	613.00	256.00	16.41	nan	nan	14.04

251	412.00	257.00	16.33	nan	15.11	14.43
252	504.00	258.00	15.14	14.72	14.44	13.97
253	383.00	260.00	15.02	14.40	13.75	13.15
254	514.00	260.00	16.31	15.65	15.11	14.51
255	318.00	263.00	15.51	14.62	14.05	13.35
256	39.00	264.00	17.22	16.27	15.88	15.38
257	484.00	264.00	15.31	14.57	13.99	13.36
258	182.00	268.00	16.35	15.51	14.88	14.12
259	529.00	268.00	14.97	14.02	13.37	12.62
260	589.00	268.00	19.43	16.76	15.80	14.77
261	597.00	268.00	nan	17.84	16.93	15.87
262	280.00	269.00	16.29	15.57	15.06	14.42
263	517.00	269.00	15.31	14.72	nan	13.64
264	325.00	270.00	16.60	15.84	15.26	14.59
265	416.00	271.00	15.03	14.74	14.50	14.06
266	485.00	273.00	14.78	nan	nan	nan
267	102.00	274.00	nan	nan	19.74	20.25
268	476.00	274.00	14.99	nan	nan	nan
269	292.00	275.00	15.23	14.91	14.73	14.35
270	267.00	276.00	17.37	16.53	15.93	15.08
271	307.00	276.00	15.78	15.07	14.56	13.87
272	434.00	276.00	14.84	14.35	nan	nan
273	460.00	277.00	16.54	15.84	15.30	14.57
274	333.00	277.00	16.75	16.06	15.36	14.50
275	349.00	279.00	14.05	nan	12.35	nan
276	16.00	280.00	15.71	15.15	14.76	14.33
277	123.00	280.00	15.55	14.64	14.00	13.28
278	608.00	283.00	17.25	16.24	15.74	15.24
279	138.00	284.00	17.68	nan	nan	nan
280	452.00	283.00	16.07	nan	14.85	14.18
281	393.00	284.00	15.94	15.30	14.59	13.81
282	544.00	285.00	16.93	16.10	15.53	14.83
283	343.00	286.00	14.43	13.55	12.90	12.17
284	732.00	286.00	nan	17.06	16.02	15.00
285	430.00	286.00	15.72	15.02	14.43	13.76
286	253.00	288.00	14.99	14.29	13.77	nan
287	301.00	291.00	16.65	15.75	15.23	14.54
288	417.00	292.00	14.62	13.71	13.08	12.31
289	476.00	292.00	13.68	12.55	11.81	10.96
290	96.00	293.00	15.14	14.96	14.88	14.77
291	214.00	296.00	17.02	16.26	15.75	14.99
292	327.00	296.00	16.61	16.02	15.41	14.69
293	339.00	295.00	15.94	nan	nan	nan
294	235.00	297.00	18.30	18.57	17.67	16.63
295	171.00	299.00	17.93	nan	nan	nan
296	379.00	299.00	14.79	nan	nan	nan
297	522.00	300.00	13.96	12.86	12.12	11.29
298	267.00	302.00	17.40	16.45	15.91	15.25

299	295.00	302.00	15.30	14.99	14.81	14.45
300	425.00	302.00	14.13	13.08	12.40	11.60
301	437.00	302.00	15.42	nan	13.87	13.12
302	687.00	302.00	17.42	16.16	15.59	14.92
303	18.00	306.00	19.52	17.44	16.95	16.32
304	227.00	305.00	17.42	17.05	16.97	16.64
305	358.00	306.00	15.09	14.79	14.44	14.00
306	454.00	309.00	15.12	14.24	13.64	12.91
307	499.00	312.00	16.50	15.80	15.27	14.60
308	388.00	311.00	16.86	16.24	15.83	15.13
309	240.00	314.00	16.60	16.14	15.87	15.27
310	536.00	316.00	15.85	15.63	15.50	15.24
311	474.00	317.00	15.96	15.16	14.54	13.83
312	358.00	318.00	13.99	12.92	12.23	11.40
313	448.00	316.00	16.01	15.26	14.49	13.84
314	520.00	321.00	17.37	nan	16.11	nan
315	172.00	321.00	17.54	16.72	16.20	15.55
316	562.00	321.00	14.20	13.13	12.40	11.58
317	217.00	322.00	15.61	15.03	14.60	14.05
318	380.00	323.00	15.88	nan	nan	nan
319	398.00	323.00	15.16	14.39	13.85	13.18
320	462.00	323.00	15.49	nan	14.11	13.55
321	114.00	324.00	16.20	16.23	16.33	16.49
322	319.00	325.00	15.37	14.65	14.16	13.51
323	428.00	326.00	13.89	12.74	12.00	11.14
324	441.00	330.00	16.58	15.74	15.13	14.31
325	623.00	330.00	17.46	16.31	15.64	14.86
326	329.00	331.00	16.41	15.69	15.23	14.58
327	674.00	331.00	15.39	15.04	14.81	14.50
328	458.00	332.00	14.44	13.42	12.73	11.93
329	38.00	334.00	15.92	15.29	14.91	14.48
330	313.00	332.00	16.29	15.84	15.32	nan
331	510.00	334.00	15.88	14.95	14.34	13.61
332	249.00	335.00	15.70	15.16	14.74	14.17
333	344.00	335.00	16.21	nan	14.84	nan
334	371.00	335.00	15.08	14.38	13.85	13.20
335	417.00	335.00	15.55	14.72	14.08	13.31
336	499.00	336.00	15.71	14.75	nan	13.38
337	597.00	336.00	17.34	16.04	15.42	14.71
338	607.00	338.00	17.75	16.57	15.99	15.18
339	20.00	340.00	15.44	14.50	13.88	13.15
340	303.00	340.00	14.93	14.58	14.34	13.96
341	468.00	340.00	13.81	12.86	12.25	11.55
342	442.00	342.00	14.83	14.39	14.10	13.58
343	290.00	345.00	17.28	16.73	16.28	15.67
344	579.00	345.00	15.65	nan	14.08	nan
345	118.00	346.00	16.96	15.90	15.19	14.47
346	700.00	347.00	nan	nan	nan	nan

347	279.00	348.00	16.74	15.93	15.44	14.75
348	589.00	349.00	15.82	14.80	14.13	13.41
349	380.00	351.00	17.11	nan	nan	15.21
350	249.00	352.00	15.43	14.83	14.35	13.76
351	50.00	353.00	16.31	15.61	15.19	14.67
352	171.00	353.00	15.87	14.98	14.33	13.58
353	59.00	354.00	16.49	15.67	15.14	14.48
354	346.00	354.00	16.96	15.96	15.39	14.65
355	418.00	359.00	14.40	13.36	12.68	11.87
356	454.00	359.00	16.30	15.54	14.94	14.24
357	293.00	360.00	16.58	nan	nan	nan
358	394.00	360.00	17.37	17.04	16.39	15.56
359	534.00	361.00	15.57	14.69	14.04	13.31
360	547.00	362.00	17.37	16.50	15.89	15.15
361	202.00	365.00	15.85	15.35	14.90	14.37
362	151.00	369.00	15.71	15.14	14.71	14.15
363	265.00	369.00	17.31	16.81	16.35	15.78
364	91.00	370.00	17.72	nan	nan	nan
365	124.00	370.00	14.88	14.03	13.42	12.69
366	407.00	370.00	16.66	15.92	15.32	14.57
367	728.00	370.00	19.74	17.48	16.83	15.93
368	179.00	372.00	16.34	15.50	14.91	14.25
369	250.00	372.00	17.14	nan	nan	nan
370	413.00	377.00	17.39	16.37	15.77	15.01
371	389.00	378.00	13.62	12.46	nan	nan
372	318.00	380.00	17.05	16.35	15.81	15.09
373	475.00	380.00	15.63	nan	nan	nan
374	401.00	381.00	15.50	14.67	14.00	13.21
375	139.00	382.00	16.93	16.16	15.52	14.83
376	149.00	382.00	17.87	17.34	16.73	15.90
377	540.00	386.00	16.90	15.99	15.36	14.71
378	551.00	385.00	17.46	16.68	16.04	15.34
379	428.00	387.00	15.41	14.70	14.18	13.56
380	389.00	388.00	15.73	nan	nan	nan
381	334.00	392.00	17.73	17.59	17.07	16.47
382	540.00	394.00	16.44	15.60	15.01	14.36
383	478.00	396.00	17.84	16.82	16.22	15.48
384	239.00	397.00	15.41	14.71	14.14	13.45
385	262.00	398.00	15.69	15.20	14.74	14.23
386	205.00	400.00	14.84	14.60	14.47	14.26
387	606.00	400.00	19.15	17.98	17.51	16.84
388	760.00	400.00	17.53	nan	nan	nan
389	295.00	402.00	18.49	17.81	17.27	16.40
390	574.00	401.00	15.39	15.11	14.92	14.66
391	420.00	403.00	15.72	14.78	14.19	13.42
392	233.00	404.00	14.88	14.55	14.35	nan
393	182.00	408.00	17.26	16.85	16.34	15.79
394	403.00	408.00	15.39	14.81	14.35	13.73

395	525.00	410.00	17.84	16.82	16.25	15.56
396	493.00	410.00	16.33	15.49	14.87	14.20
397	591.00	410.00	16.19	16.10	16.06	16.16
398	306.00	411.00	18.17	18.08	17.60	17.28
399	385.00	414.00	15.88	15.10	14.51	13.81
400	701.00	414.00	17.79	17.01	16.40	15.62
401	203.00	415.00	15.63	14.97	14.51	13.90
402	532.00	417.00	17.98	17.17	16.47	15.84
403	328.00	417.00	15.38	14.82	14.36	13.78
404	291.00	419.00	15.31	15.12	15.02	14.86
405	485.00	419.00	14.72	13.74	13.08	12.32
406	456.00	424.00	17.99	16.87	16.20	15.42
407	469.00	424.00	16.61	15.64	15.03	14.27
408	41.00	426.00	16.57	15.99	15.52	15.04
409	441.00	427.00	16.81	15.95	15.38	14.69
410	355.00	431.00	16.26	nan	15.10	nan
411	407.00	430.00	13.92	12.78	12.02	11.16
412	155.00	431.00	15.01	14.63	14.29	13.93
413	371.00	431.00	16.16	15.46	14.89	14.23
414	422.00	431.00	17.52	16.88	16.07	15.33
415	492.00	431.00	18.05	17.00	16.35	15.51
416	581.00	432.00	16.52	15.87	15.43	14.99
417	482.00	434.00	17.50	16.55	15.84	15.11
418	177.00	434.00	20.07	19.86	18.28	17.06
419	225.00	435.00	16.73	15.87	15.24	14.53
420	248.00	435.00	16.43	15.82	15.34	14.77
421	132.00	437.00	14.24	13.17	12.46	11.62
422	389.00	440.00	16.70	16.14	15.53	nan
423	533.00	441.00	16.67	15.49	14.68	13.94
424	589.00	441.00	nan	19.10	17.87	16.61
425	537.00	451.00	15.91	15.27	14.81	14.27
426	167.00	452.00	18.55	18.99	18.76	18.02
427	269.00	453.00	15.31	15.08	14.91	14.65
428	284.00	454.00	16.00	15.19	14.58	13.90
429	563.00	459.00	17.78	16.98	16.25	15.54
430	511.00	461.00	18.76	nan	16.96	16.31
431	460.00	462.00	18.25	17.70	17.10	16.64
432	469.00	462.00	18.72	17.99	17.05	16.38
433	281.00	464.00	16.92	16.32	15.68	15.01
434	333.00	467.00	15.69	nan	14.80	14.32
435	300.00	469.00	15.40	14.86	14.43	13.90
436	353.00	477.00	15.09	14.87	14.69	14.39
437	396.00	477.00	16.65	nan	nan	nan
438	650.00	478.00	16.98	16.05	15.42	14.48
439	620.00	485.00	13.83	12.55	11.69	10.77
440	449.00	491.00	17.61	16.90	16.26	15.60
441	564.00	491.00	15.48	14.93	14.53	13.98
442	640.00	492.00	14.93	13.43	12.14	10.00

443	274.00	497.00	15.91	15.21	14.64	13.89
444	460.00	500.00	15.15	nan	14.36	12.99
445	315.00	502.00	15.17	14.44	nan	13.01
446	493.00	504.00	18.55	nan	nan	nan
447	716.00	4.00	nan	17.71	16.22	14.86
448	698.00	11.00	nan	nan	nan	nan
449	96.00	12.00	nan	nan	nan	nan
450	424.00	13.00	nan	19.26	16.86	15.76
451	547.00	14.00	nan	nan	nan	nan
452	205.00	15.00	nan	19.10	17.67	16.93
453	401.00	18.00	nan	14.12	nan	nan
454	484.00	18.00	nan	nan	nan	nan
455	625.00	18.00	nan	nan	nan	nan
456	297.00	22.00	nan	nan	nan	nan
457	329.00	22.00	nan	21.15	18.36	17.65
458	245.00	24.00	nan	nan	nan	nan
459	364.00	25.00	nan	18.91	nan	nan
460	224.00	25.00	nan	20.52	17.72	16.73
461	436.00	25.00	nan	15.84	15.07	14.34
462	480.00	26.00	nan	nan	nan	nan
463	655.00	29.00	nan	nan	nan	nan
464	232.00	33.00	nan	nan	nan	nan
465	599.00	33.00	nan	nan	nan	nan
466	145.00	36.00	nan	nan	nan	nan
467	317.00	38.00	nan	nan	nan	nan
468	448.00	43.00	nan	nan	nan	nan
469	406.00	47.00	nan	18.62	17.48	16.85
470	663.00	46.00	nan	nan	nan	nan
471	314.00	47.00	nan	nan	nan	nan
472	488.00	47.00	nan	nan	nan	nan
473	459.00	48.00	nan	18.69	17.17	16.39
474	282.00	49.00	nan	nan	nan	nan
475	328.00	50.00	nan	17.36	nan	nan
476	578.00	50.00	nan	nan	18.03	17.10
477	354.00	55.00	nan	19.96	17.79	16.96
478	205.00	56.00	nan	nan	21.64	17.75
479	332.00	58.00	nan	16.56	15.99	15.31
480	114.00	60.00	nan	18.29	17.16	16.34
481	220.00	66.00	nan	16.75	14.47	nan
482	559.00	66.00	nan	19.45	17.83	17.29
483	358.00	67.00	nan	18.12	17.57	16.56
484	407.00	68.00	nan	16.89	nan	nan
485	635.00	68.00	nan	nan	nan	nan
486	16.00	69.00	nan	nan	nan	nan
487	442.00	70.00	nan	17.20	nan	nan
488	345.00	70.00	nan	17.95	17.80	16.87
489	376.00	70.00	nan	17.81	16.96	16.46
490	397.00	73.00	nan	18.18	nan	nan

491	199.00	73.00	nan	nan	nan	18.23
492	548.00	74.00	nan	18.00	17.01	16.24
493	151.00	74.00	nan	nan	nan	nan
494	305.00	75.00	nan	16.77	16.00	15.50
495	189.00	75.00	nan	nan	nan	nan
496	507.00	76.00	nan	16.75	nan	nan
497	289.00	81.00	nan	nan	18.32	nan
498	135.00	81.00	nan	nan	nan	nan
499	527.00	81.00	nan	nan	nan	nan
500	544.00	81.00	nan	18.66	nan	nan
501	343.00	82.00	nan	19.13	nan	nan
502	409.00	82.00	nan	19.10	17.28	15.43
503	714.00	83.00	nan	nan	nan	nan
504	385.00	85.00	nan	16.32	nan	nan
505	470.00	88.00	nan	18.18	17.31	16.40
506	137.00	91.00	nan	nan	nan	nan
507	153.00	91.00	nan	nan	nan	nan
508	448.00	93.00	nan	16.92	nan	15.25
509	651.00	95.00	nan	nan	17.82	16.80
510	310.00	97.00	nan	17.05	16.47	15.87
511	378.00	98.00	nan	19.44	17.82	16.73
512	63.00	101.00	nan	nan	nan	nan
513	346.00	102.00	nan	17.22	16.47	15.75
514	511.00	105.00	nan	18.04	nan	16.35
515	93.00	110.00	nan	16.80	16.26	15.59
516	121.00	110.00	nan	nan	nan	nan
517	206.00	113.00	nan	17.40	16.82	16.12
518	472.00	114.00	nan	17.22	16.58	15.82
519	331.00	116.00	nan	17.29	nan	nan
520	260.00	119.00	nan	15.83	15.29	14.63
521	268.00	120.00	nan	14.73	14.27	13.69
522	582.00	120.00	nan	18.80	nan	nan
523	429.00	123.00	nan	16.88	16.19	15.44
524	480.00	123.00	nan	18.10	nan	16.45
525	529.00	123.00	nan	17.31	16.51	15.68
526	441.00	126.00	nan	17.16	nan	15.68
527	332.00	128.00	nan	16.36	15.60	14.97
528	499.00	126.00	nan	18.61	17.71	16.65
529	110.00	133.00	nan	nan	nan	nan
530	159.00	131.00	nan	15.75	15.20	14.50
531	171.00	132.00	nan	nan	nan	nan
532	427.00	132.00	nan	17.34	nan	15.59
533	364.00	134.00	nan	16.37	nan	nan
534	416.00	135.00	nan	16.59	16.01	15.25
535	487.00	133.00	nan	16.62	15.99	15.25
536	570.00	134.00	nan	nan	nan	nan
537	88.00	135.00	nan	19.83	nan	nan
538	441.00	136.00	nan	16.39	15.61	14.87

539	660.00	139.00	nan	nan	nan	nan
540	504.00	140.00	nan	19.64	18.45	16.96
541	189.00	142.00	nan	nan	nan	nan
542	617.00	143.00	nan	nan	nan	nan
543	122.00	144.00	nan	nan	nan	nan
544	365.00	144.00	nan	15.79	15.17	14.52
545	216.00	145.00	nan	18.41	18.84	nan
546	537.00	149.00	nan	16.97	16.46	15.68
547	593.00	150.00	nan	18.75	17.60	16.98
548	7.00	156.00	nan	nan	nan	nan
549	135.00	158.00	nan	nan	nan	19.82
550	505.00	159.00	nan	17.67	nan	16.00
551	209.00	161.00	nan	18.63	nan	nan
552	702.00	162.00	nan	18.89	17.70	17.34
553	111.00	163.00	nan	17.56	17.10	16.36
554	315.00	164.00	nan	14.25	nan	nan
555	612.00	164.00	nan	17.96	17.21	16.68
556	494.00	165.00	nan	16.87	16.36	15.56
557	258.00	168.00	nan	16.72	16.36	15.60
558	352.00	169.00	nan	16.62	nan	nan
559	201.00	169.00	nan	20.69	18.73	17.55
560	579.00	168.00	nan	20.27	nan	nan
561	309.00	170.00	nan	14.73	14.24	13.62
562	273.00	173.00	nan	17.90	nan	16.12
563	488.00	173.00	nan	16.67	16.08	15.40
564	736.00	173.00	nan	nan	nan	nan
565	709.00	175.00	nan	nan	nan	nan
566	131.00	177.00	nan	nan	nan	nan
567	499.00	178.00	nan	16.34	15.71	14.97
568	366.00	179.00	nan	15.50	15.01	14.38
569	592.00	182.00	nan	17.18	16.37	15.53
570	615.00	182.00	nan	nan	nan	nan
571	542.00	183.00	nan	16.54	15.82	15.10
572	144.00	185.00	nan	20.15	19.66	18.07
573	408.00	185.00	nan	16.66	15.95	15.13
574	505.00	186.00	nan	16.51	15.90	nan
575	362.00	189.00	nan	15.41	nan	nan
576	481.00	189.00	nan	16.51	nan	nan
577	658.00	190.00	nan	nan	nan	nan
578	411.00	193.00	nan	16.66	nan	nan
579	541.00	193.00	nan	16.85	16.24	15.55
580	247.00	193.00	nan	17.33	17.02	16.09
581	118.00	197.00	nan	nan	nan	nan
582	685.00	202.00	nan	17.69	nan	nan
583	341.00	205.00	nan	15.66	nan	nan
584	636.00	206.00	nan	17.50	16.71	16.24
585	509.00	207.00	nan	nan	nan	17.75
586	250.00	208.00	nan	17.74	nan	16.51

587	406.00	208.00	nan	13.43	12.78	12.00
588	488.00	211.00	nan	16.72	nan	15.48
589	496.00	213.00	nan	17.02	nan	nan
590	567.00	215.00	nan	17.11	16.75	16.19
591	213.00	217.00	nan	16.12	15.56	14.75
592	268.00	217.00	nan	17.13	16.53	15.73
593	482.00	220.00	nan	16.75	16.29	nan
594	142.00	224.00	nan	nan	nan	nan
595	377.00	224.00	nan	15.62	15.10	14.44
596	470.00	225.00	nan	16.96	nan	nan
597	603.00	229.00	nan	nan	nan	nan
598	266.00	234.00	nan	16.43	16.02	nan
599	7.00	236.00	nan	nan	nan	18.99
600	357.00	237.00	nan	15.66	15.08	14.41
601	205.00	237.00	nan	16.57	nan	nan
602	156.00	237.00	nan	nan	nan	nan
603	558.00	241.00	nan	17.10	16.43	nan
604	726.00	244.00	nan	nan	nan	nan
605	701.00	245.00	nan	nan	nan	nan
606	484.00	247.00	nan	16.54	15.96	15.12
607	475.00	248.00	nan	16.37	15.87	nan
608	520.00	252.00	nan	16.50	nan	nan
609	539.00	257.00	nan	17.16	nan	nan
610	431.00	258.00	nan	16.59	16.00	15.29
611	117.00	259.00	nan	nan	nan	nan
612	176.00	259.00	nan	17.92	17.23	16.21
613	8.00	262.00	nan	nan	nan	nan
614	397.00	262.00	nan	15.79	15.26	14.57
615	294.00	264.00	nan	16.43	15.91	15.28
616	364.00	264.00	nan	16.27	nan	nan
617	540.00	267.00	nan	16.46	15.66	14.78
618	66.00	269.00	nan	nan	nan	nan
619	158.00	272.00	nan	nan	nan	nan
620	405.00	271.00	nan	15.92	nan	nan
621	552.00	273.00	nan	18.82	nan	nan
622	512.00	277.00	nan	16.09	nan	14.92
623	620.00	284.00	nan	17.94	nan	16.77
624	324.00	285.00	nan	16.73	16.23	15.53
625	366.00	284.00	nan	16.27	nan	15.06
626	442.00	289.00	nan	16.13	15.58	14.86
627	531.00	288.00	nan	17.78	16.96	15.97
628	599.00	291.00	nan	nan	nan	18.31
629	676.00	293.00	nan	nan	19.02	18.67
630	316.00	293.00	nan	16.94	nan	nan
631	541.00	297.00	nan	17.58	nan	16.23
632	732.00	298.00	nan	nan	nan	nan
633	141.00	301.00	nan	18.30	17.67	17.20
634	508.00	301.00	nan	16.48	nan	nan

635	247.00	304.00	nan	19.18	nan	17.07
636	658.00	306.00	nan	nan	nan	nan
637	313.00	309.00	nan	17.80	nan	nan
638	484.00	308.00	nan	16.61	16.04	15.27
639	135.00	309.00	nan	nan	nan	nan
640	376.00	311.00	nan	16.60	15.94	15.15
641	305.00	312.00	nan	17.59	17.21	16.37
642	554.00	313.00	nan	16.82	15.55	14.58
643	429.00	317.00	nan	15.12	nan	nan
644	283.00	317.00	nan	17.58	17.16	16.25
645	295.00	317.00	nan	17.63	17.06	16.35
646	415.00	318.00	nan	16.25	15.59	14.66
647	704.00	320.00	nan	nan	nan	nan
648	185.00	322.00	nan	17.11	16.41	15.68
649	601.00	324.00	nan	18.28	17.50	16.77
650	105.00	327.00	nan	18.05	18.47	17.89
651	228.00	327.00	nan	17.65	17.29	nan
652	288.00	329.00	nan	17.40	16.89	16.11
653	299.00	331.00	nan	16.83	16.20	15.58
654	715.00	331.00	nan	nan	nan	nan
655	205.00	335.00	nan	18.06	17.69	16.71
656	233.00	339.00	nan	nan	nan	nan
657	747.00	339.00	nan	17.78	17.47	16.85
658	376.00	343.00	nan	16.56	nan	nan
659	494.00	343.00	nan	16.12	nan	nan
660	715.00	343.00	nan	nan	nan	nan
661	401.00	347.00	nan	16.96	nan	nan
662	509.00	350.00	nan	17.87	17.54	16.58
663	537.00	352.00	nan	16.34	15.50	15.10
664	324.00	354.00	nan	17.29	nan	nan
665	370.00	353.00	nan	17.19	nan	15.92
666	550.00	353.00	nan	nan	nan	nan
667	615.00	354.00	nan	19.54	17.98	17.08
668	124.00	355.00	nan	17.71	16.99	16.21
669	730.00	356.00	nan	nan	nan	nan
670	601.00	357.00	nan	nan	nan	nan
671	268.00	358.00	nan	17.45	17.01	16.44
672	310.00	359.00	nan	16.89	nan	15.71
673	696.00	359.00	nan	nan	nan	nan
674	642.00	360.00	nan	19.86	18.17	17.50
675	44.00	362.00	nan	nan	nan	nan
676	319.00	371.00	nan	16.69	16.05	15.39
677	357.00	374.00	nan	17.18	16.79	16.11
678	457.00	373.00	nan	17.59	nan	nan
679	557.00	374.00	nan	18.77	nan	nan
680	447.00	375.00	nan	17.10	16.32	15.47
681	235.00	376.00	nan	18.03	17.78	17.12
682	273.00	376.00	nan	18.83	17.67	16.84

683	624.00	378.00	nan	20.53	18.81	19.71
684	91.00	380.00	nan	19.22	18.02	16.90
685	202.00	381.00	nan	17.25	nan	16.41
686	585.00	382.00	nan	nan	19.50	18.87
687	515.00	383.00	nan	18.16	17.40	nan
688	295.00	385.00	nan	21.38	nan	nan
689	464.00	391.00	nan	16.84	nan	nan
690	622.00	392.00	nan	nan	nan	nan
691	287.00	393.00	nan	17.98	17.84	17.08
692	665.00	394.00	nan	20.94	19.86	18.76
693	409.00	396.00	nan	17.97	nan	nan
694	449.00	400.00	nan	18.61	nan	nan
695	127.00	402.00	nan	nan	nan	nan
696	109.00	403.00	nan	nan	nan	nan
697	19.00	405.00	nan	nan	nan	nan
698	721.00	405.00	nan	nan	nan	nan
699	250.00	407.00	nan	17.41	16.73	15.91
700	258.00	410.00	nan	17.51	16.90	16.20
701	575.00	411.00	nan	21.22	nan	nan
702	626.00	414.00	nan	nan	nan	nan
703	565.00	415.00	nan	19.72	21.72	nan
704	515.00	416.00	nan	17.89	17.43	16.76
705	748.00	416.00	nan	19.06	18.59	nan
706	101.00	417.00	nan	nan	nan	nan
707	657.00	424.00	nan	nan	nan	nan
708	543.00	428.00	nan	18.23	nan	nan
709	526.00	429.00	nan	nan	nan	nan
710	297.00	431.00	nan	nan	nan	nan
711	276.00	433.00	nan	22.15	20.79	19.68
712	669.00	435.00	nan	nan	nan	nan
713	284.00	438.00	nan	nan	nan	nan
714	743.00	441.00	nan	17.26	16.88	16.39
715	645.00	445.00	nan	nan	nan	nan
716	293.00	447.00	nan	18.22	17.54	16.99
717	390.00	448.00	nan	18.77	nan	nan
718	308.00	454.00	nan	nan	nan	nan
719	333.00	455.00	nan	18.24	17.65	17.13
720	617.00	455.00	nan	18.83	17.77	16.88
721	153.00	458.00	nan	nan	nan	nan
722	413.00	465.00	nan	nan	nan	nan
723	67.00	466.00	nan	nan	nan	nan
724	180.00	467.00	nan	nan	nan	nan
725	495.00	468.00	nan	18.67	18.34	17.81
726	113.00	472.00	nan	nan	nan	nan
727	485.00	474.00	nan	17.76	nan	nan
728	195.00	474.00	nan	20.06	19.57	20.06
729	428.00	474.00	nan	nan	nan	nan
730	659.00	476.00	nan	17.94	17.20	16.09

731	503.00	478.00	nan	nan	nan	nan
732	526.00	479.00	nan	nan	nan	nan
733	604.00	478.00	nan	nan	nan	18.25
734	722.00	480.00	nan	nan	nan	nan
735	445.00	483.00	nan	17.34	nan	nan
736	429.00	488.00	nan	nan	20.67	18.86
737	658.00	492.00	nan	nan	19.12	16.18
738	126.00	493.00	nan	18.35	17.55	16.74
739	28.00	497.00	nan	nan	nan	nan
740	292.00	505.00	nan	20.41	nan	nan
741	318.00	7.00	nan	nan	nan	nan
742	190.00	9.00	nan	nan	nan	nan
743	393.00	9.00	nan	nan	17.94	17.14
744	405.00	19.00	nan	nan	13.81	13.40
745	260.00	30.00	nan	nan	16.50	nan
746	354.00	39.00	nan	nan	nan	nan
747	438.00	45.00	nan	nan	16.90	16.26
748	567.00	51.00	nan	nan	18.29	nan
749	594.00	91.00	nan	nan	17.77	nan
750	357.00	109.00	nan	nan	15.73	nan
751	653.00	117.00	nan	nan	16.33	15.67
752	196.00	122.00	nan	nan	22.55	17.98
753	464.00	124.00	nan	nan	17.53	nan
754	209.00	127.00	nan	nan	17.18	16.42
755	67.00	145.00	nan	nan	11.04	10.24
756	198.00	151.00	nan	nan	17.39	16.67
757	388.00	155.00	nan	nan	13.40	12.63
758	381.00	163.00	nan	nan	14.72	nan
759	410.00	165.00	nan	nan	14.17	13.56
760	677.00	164.00	nan	nan	nan	nan
761	345.00	172.00	nan	nan	15.94	nan
762	262.00	180.00	nan	nan	16.28	nan
763	754.00	186.00	nan	nan	18.50	16.51
764	237.00	190.00	nan	nan	16.74	nan
765	458.00	216.00	nan	nan	11.95	11.12
766	575.00	219.00	nan	nan	16.39	15.85
767	205.00	227.00	nan	nan	14.54	13.90
768	340.00	248.00	nan	nan	11.48	nan
769	454.00	250.00	nan	nan	13.94	nan
770	228.00	261.00	nan	nan	17.81	nan
771	438.00	265.00	nan	nan	15.39	14.75
772	632.00	270.00	nan	nan	19.50	18.88
773	205.00	273.00	nan	nan	16.98	16.15
774	562.00	275.00	nan	nan	16.87	16.15
775	457.00	291.00	nan	nan	15.38	14.60
776	337.00	294.00	nan	nan	14.65	14.01
777	405.00	297.00	nan	nan	12.79	nan
778	508.00	298.00	nan	nan	15.85	nan

779	751.00	308.00	nan	nan	nan	nan
780	524.00	328.00	nan	nan	16.19	nan
781	385.00	333.00	nan	nan	15.57	14.83
782	478.00	333.00	nan	nan	15.42	14.70
783	429.00	356.00	nan	nan	15.15	14.35
784	422.00	368.00	nan	nan	14.97	13.88
785	159.00	387.00	nan	nan	17.33	16.65
786	500.00	400.00	nan	nan	16.97	16.36
787	312.00	429.00	nan	nan	17.00	nan
788	687.00	446.00	nan	nan	15.91	15.28
789	353.00	455.00	nan	nan	16.52	nan
790	426.00	459.00	nan	nan	nan	nan
791	628.00	461.00	nan	nan	18.12	nan
792	572.00	483.00	nan	nan	nan	nan
793	407.00	492.00	nan	nan	16.61	nan
794	475.00	12.00	nan	nan	nan	17.84
795	755.00	38.00	nan	nan	nan	17.89
796	696.00	61.00	nan	nan	nan	nan
797	221.00	69.00	nan	nan	nan	12.65
798	335.00	84.00	nan	nan	nan	15.71
799	573.00	108.00	nan	nan	nan	12.61
800	143.00	115.00	nan	nan	nan	12.02
801	603.00	141.00	nan	nan	nan	nan
802	425.00	154.00	nan	nan	nan	11.35
803	481.00	153.00	nan	nan	nan	14.80
804	342.00	183.00	nan	nan	nan	14.19
805	423.00	190.00	nan	nan	nan	14.65
806	592.00	194.00	nan	nan	nan	12.85
807	548.00	218.00	nan	nan	nan	18.37
808	314.00	351.00	nan	nan	nan	15.98
809	380.00	360.00	nan	nan	nan	14.82
810	343.00	365.00	nan	nan	nan	15.71
811	437.00	372.00	nan	nan	nan	12.80
812	4.00	396.00	nan	nan	nan	14.96
813	378.00	423.00	nan	nan	nan	13.80
814	357.00	432.00	nan	nan	nan	14.47
815	604.00	458.00	nan	nan	nan	nan
816	312.00	493.00	nan	nan	nan	15.63
817	432.00	503.00	nan	nan	nan	16.87
818	266.00	504.00	nan	nan	nan	15.62
819	383.00	504.00	nan	nan	nan	15.92

1.14 Check difference in magnitudes of reference stars from photometric and non-photometric images

Ok, now for each of five photometric stars, I have magnitudes measured both from photometric and non-photometric images. These measurements should be similar. Let's check that this is true.

[12]:
```
def verify_reference_magnitudes(data_dir, filter_names,␣
↪max_magnitude_difference):
    # Read the magnitudes from CSV files
    # ------

    df_non_photometric = pd.read_csv(os.path.join(data_dir, "magnitudes.csv"))

    df_photometric = pd.read_csv(os.path.join(data_dir,␣
↪"reference_stars_magnitudes.csv"),
                                 index_col="star_number")

    positions = df_non_photometric[["x", "y"]].values.tolist()
    max_star_distance = 2

    # Create a table to store magnitude differences between photometric and␣
↪non-photometric images
    # ------

    delta_column_names = [f'{name.lower()}_delta' for name in filter_names]
    diff_column_names = ["x", "y"] + delta_column_names


    df_delta = pd.DataFrame(columns=diff_column_names, index=df_photometric.
↪index)
    df_delta.index.name = "star_number"

    # Iterate over reference stars in photometric table
    for star_number, row_photometric in df_photometric.iterrows():
        # Get x-y coordinates of the star
        x_from_photometric = row_photometric['non_photometric_x']
        y_from_photometric = row_photometric['non_photometric_y']
        position = (x_from_photometric, y_from_photometric)

        # Find the position of this star in non-photometric table
        star_index = find_star_index(positions=positions,
                                     distance=max_star_distance,
                                     point=position)

        if star_index is None:
            raise RuntimeError(f"ERROR: Can't find reference star {position}")

        # Get the table row for the star from non_photometric table
        row_non_photometric = df_non_photometric.iloc[star_index]

        # List of column names for all magnitudes
        column_names = [f'{name.lower()}_mag' for name in filter_names]
```

```
        # Calculate difference of magnitudes betwen the photometric
        # and non-photometric images

        delta_mags = [
            row_photometric[column_name] - row_non_photometric[column_name]
            for column_name in column_names
        ]

        # Store the star's position and magnitude differences in the table
        # -------

        df_delta.loc[star_number]['x'] = x_from_photometric
        df_delta.loc[star_number]['y'] = y_from_photometric

        for filter_name, delta_mag in zip(filter_names, delta_mags):
            # Raise error if magnitude difference is too large
            if abs(delta_mag) > max_magnitude_difference:
                raise RuntimeError((
                    f"ERROR: magnitude difference {delta_mag:.3f} for "
                    f"reference star {star_number} is too lage"
                ))

            df_delta.loc[star_number][f'{filter_name.lower()}_delta'] =␣
↪round(delta_mag, 2)

    # Save coordinates as integers, not floats
    df_delta.x = df_delta.x.astype(int)
    df_delta.y = df_delta.y.astype(int)

    # Print the table of magnitude differences
    print("\nTable 5: Differences of magnitudes of reference stars between␣
↪photometric and non-photometric images.\n")

    print(tabulate(df_delta,
                   headers=["#"] + diff_column_names,
                   floatfmt=".2f", tablefmt="github"))

    # Calculate mean and standard deviation of magnitude differences
    # --------

    print("\n\nTable 6: Mean standard deviations of differences of magnitudes of␣
↪reference stars between photometric and non-photometric images.")
    print("----------------------")

    for filter_name in filter_names:
        column_name = f'{filter_name.lower()}_delta'
```

```
        print(f"{filter_name}: {df_delta[column_name].mean():>7.2f} ±␣
↪{df_delta[column_name].std():.2f} mag")


    # Save magnitudes to a CSV file
    file_path = os.path.join(data_dir, "magnitude_differences.csv")
    df_delta.to_csv(file_path, index=False)
    print(f'\nMagnitude differences saved to {file_path}')


verify_reference_magnitudes(data_dir=data_dir, filter_names=filter_names,␣
↪max_magnitude_difference=0.2)
```

Table 5: Differences of magnitudes of reference stars between photometric and non-photometric images.

#	x	y	b_delta	v_delta	r_delta	i_delta
1	48	27	-0.12	-0.03	0.01	-0.02
2	293	56	-0.06	-0.02	-0.03	-0.04
3	123	370	0.08	-0.04	-0.05	-0.02
4	328	417	0.11	0.02	0.04	0.01
5	653	245	0.00	0.07	0.02	0.07

Table 6: Mean standard deviations of differences of magnitudes of reference stars between photometric and non-photometric images.

```
---------------------
B:    0.00 ± 0.10 mag
V:    0.00 ± 0.05 mag
R:   -0.00 ± 0.04 mag
I:    0.00 ± 0.04 mag

Magnitude differences saved to data/magnitude_differences.csv
```

1.14.1 Are my magnitude differences any good?

From Table 6 I can see that standard deviation of magnitude differences is 0.04 mag for all filter excep B, where it is 0.11 mag. I don't know it these value are good or bad, will ask my teachers. These numbers are similar to standard deviations of magnitudes from 5 measurements I calculated earlier in Table 3. The magnitudes are more different for B filter probably because I only used one photometric B image.

1.15 Plot color magnitude diagrams

As another check I plot a color-magnitude diagram. I expect to see the main sequence that turns into a red giant branch. Let's see if that's what's up.

[13]:
```
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)



def part_a_make_cmd(plot_dir, file_name, data_path, blue_mag, red_mag,
                    x_label, y_label, title, xlims, ylims):
    """
    Make a plot of colour magnitude diagram.

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

    file_name: str
        Plot file name.

    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.
    """

    # 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, df[red_mag], zorder=2,
               color="#0084ff40",
               edgecolor="#0084ff")

    # 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)
```

```
    ax.set_title(title, y=-0.15)  # Set image title

    # 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)


data_path = os.path.join(data_dir, "magnitudes.csv")

all_plot_settings = [
    dict(magnitudes=["B", "V"], xlims=(-0.5, 3), ylims=(11.5, 20)),
    dict(magnitudes=["B", "R"], xlims=(-0.5, 3)),
    dict(magnitudes=["B", "I"], xlims=(-0.5, 5)),
    dict(magnitudes=["V", "R"], xlims=(-0.5, 2)),
    dict(magnitudes=["V", "I"], xlims=(-0.5, 4), ylims=(10, 20)),
    dict(magnitudes=["R", "I"], xlims=(-0.5, 2), ylims=(10, 20))
]

figure_number = 10

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

    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."
    )
```

```
   part_a_make_cmd(plot_dir="images",␣
↪file_name=f"cmd_{blue_mag_lowcase}_{red_mag_lowcase}.png",
                   data_path=data_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)
```

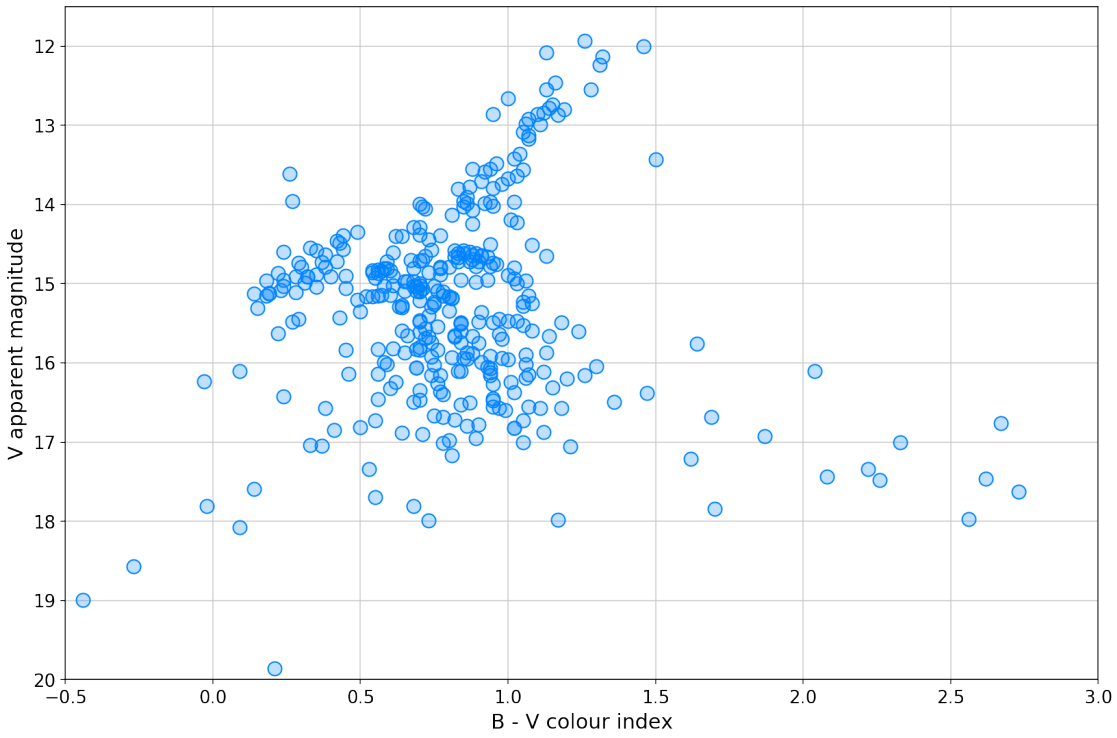


Figure 11: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

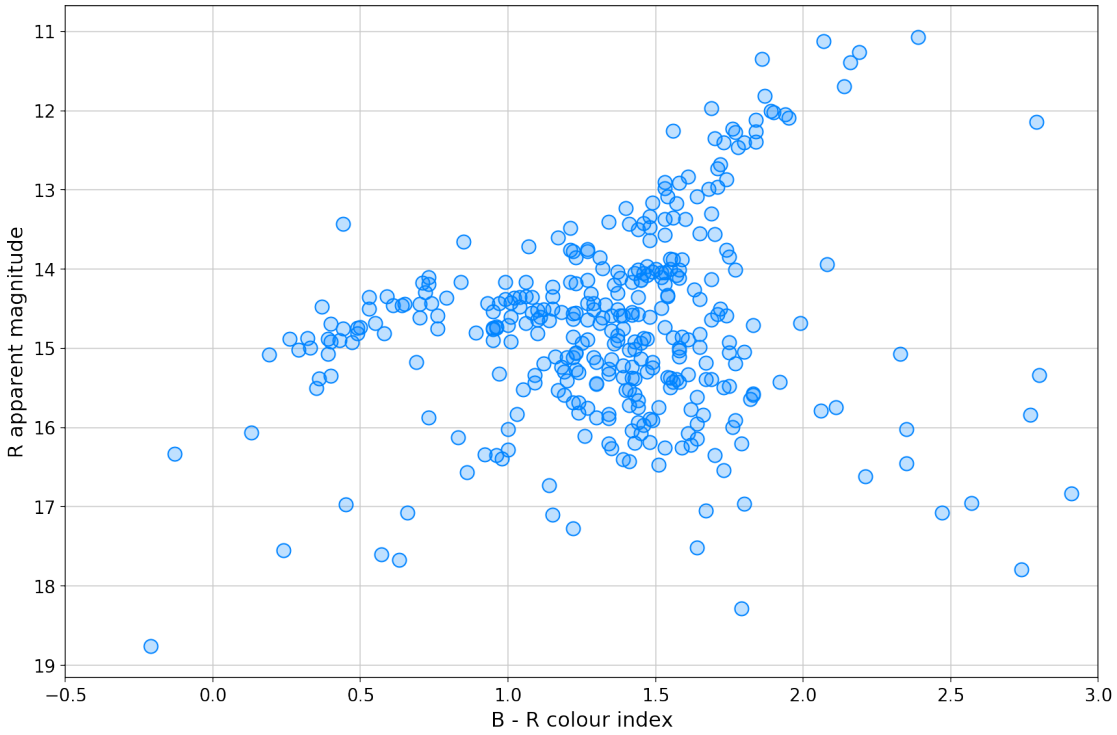


Figure 12: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

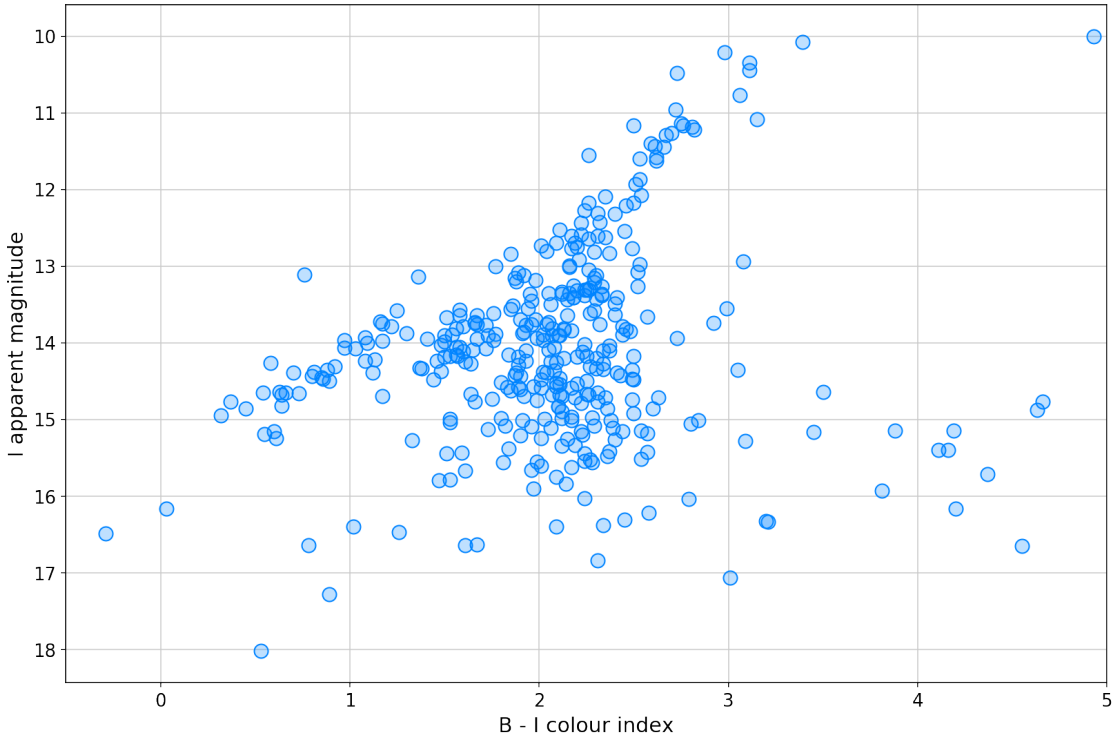


Figure 13: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

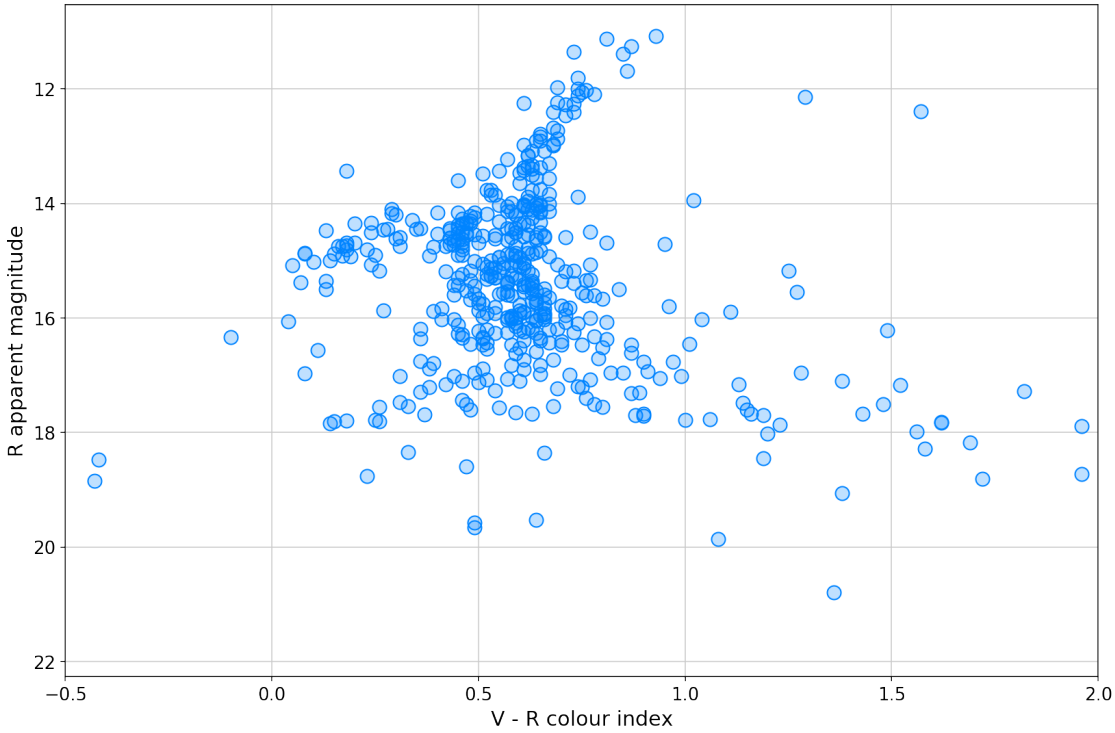


Figure 14: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

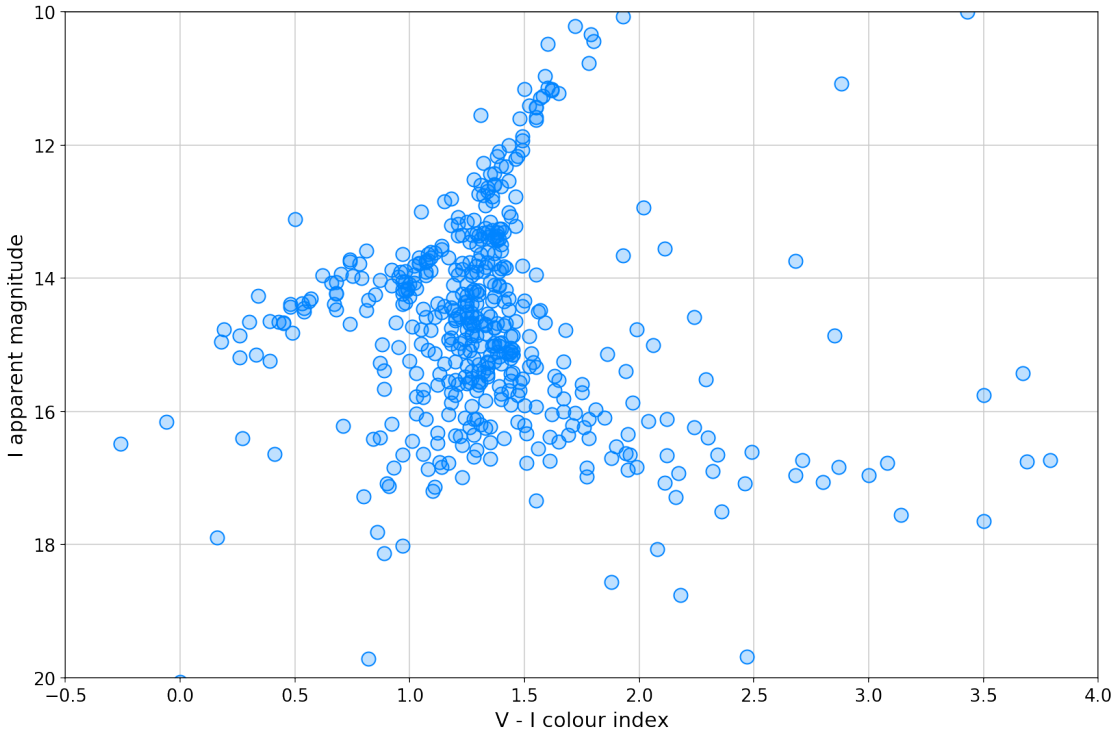


Figure 15: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

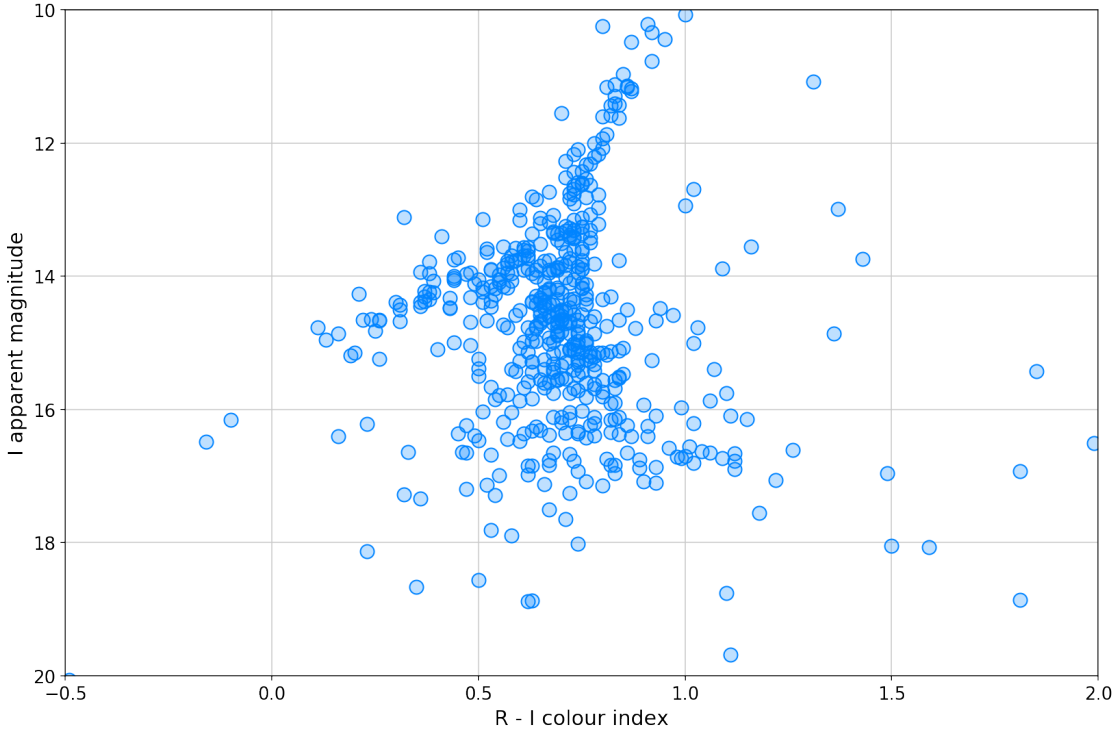


Figure 16: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

1.16 Do color-magnitude diagrams make sense?

No

[14]:
```
print("We are done!")
```

```
We are done!
```

[ ]:
```

```