

Optimized Curve Fitting (OCF) Algorithm

Deval Deliwala

GOAL

The Optimized Curve Fitting (OCF) algorithm aims to determine the slope of the Red Clump (RC) cluster in a color-magnitude diagram (CMD) with minimal manual intervention. Traditional methods often rely on visual adjustments and unsharp masking techniques to identify the RC slope, leading to potential inconsistencies and biases. The OCF algorithm provides a systematic and reproducible approach to identifying the RC slope by segmenting the RC bar into defined tiles and iteratively fitting compound models on the extracted star data.

1 Procedure

1.1 Inputs & Parameters

The algorithm requires the following inputs:

1. **Star Catalogs:**

Two matched star catalogs, designated as `catalog1` and `catalog2`, are used to generate the CMD. The CMD is plotted as `catalog1 - catalog2` vs. `catalogy`, where `catalogy` is either `catalog1` or `catalog2`.

2. **Initial Parallel Cutoffs:**

A set of parallel lines that *roughly* delineate the RC slope. These lines do not need to be highly precise but should tightly bound the RC bar.

3. **x -Range Cutoff:**

The range in the x -axis color space that defines the width of the RC bar.

4. **Tile Count n :**

An integer specifying the number of segments/tiles to divide the RC bar.

The provided inputs for an example CMD are visually represented in Figure 1, illustrating the parallel and x -range cutoff used to bound the RC cluster.

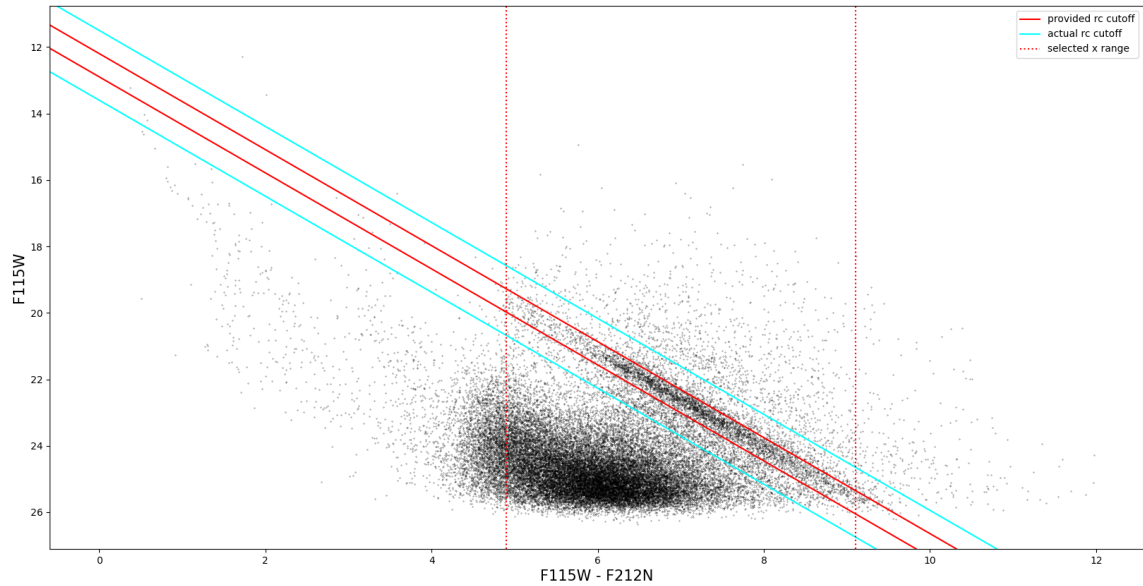


Figure 1: Example NRCB1 $F115W - F212N$ vs. $F115W$ provided cutoffs

1.2 Preliminary Steps

1.2.1 Extend Parallel Cutoffs

The algorithm begins by extending the width of the initial parallel lines by a factor of 3, allowing for a broader range of stars to be included in the analysis. The expanded lines are also shown in Figure 1 in blue.

1.2.2 Generate the First Tile

Using the extended parallel lines and the specified tile count n , the algorithm generates a *starting tile* of width x_range / n . It is placed at the left-most end of the provided x_range , in between the parallel cutoffs, extending horizontally by the calculated tile width in a left-riemann fashion. An example starting bin for $n = 9$ is shown in Figure 2 below.

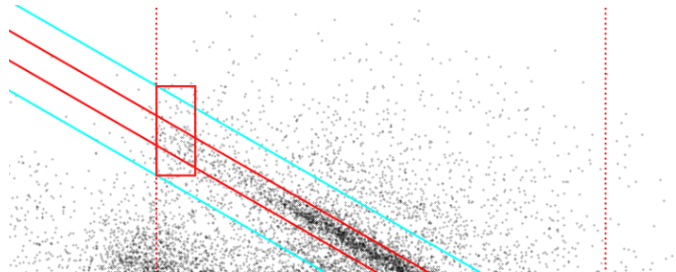


Figure 2: Starting bin for $n = 9$

1.2.3 Extract Stars from Starting Tile

The algorithm then extracts the indices of stars that fall within the bounds of the first tile. The corresponding magnitudes from `catalog1` and `catalog2` are retrieved and these stars are compiled into a pandas DataFrame object for subsequent analysis.

1.3 OCF Algorithm

Once the magnitudes of the stars in the starting bin have been extracted, the algorithm proceeds to the OCF phase for this specific bin. The primary goal of OCF is to produce a compound model using the `astropy.models` library that fits the histogram of `catalogy` magnitude data optimally, thereby minimizing the error of the mean magnitude for stars in that bin.

The compound model used consists of a Gaussian component (`Gaussian1D()`) and a linear component (`Linear1D()`). This combination is chosen because it allows the mean magnitude to be calculated more accurately by focusing the Gaussian fit on the dense region of stars, while the linear component accounts for outliers and stars outside the dense region.

1.3.1 Compound Model Parameters

A compound `Gaussian1D() + Linear1D()` model requires five parameters:

1. Gaussian Component:
 - Amplitude
 - Mean
 - Standard Deviation
2. Linear Component:
 - Slope
 - Intercept

The objective of OCF is to determine the optimal values for these parameters for a compound model such that the error in the model's returned mean parameter is minimized.

1.3.2 OCF Steps

The OCF Algorithm employs two main iterative steps to achieve optimal fitting:

1. **Looser-Requirement Iterative Compound Fitting:**
 - This step aims to determine suitable initial parameter values for the Gaussian mean and standard deviation. It uses the overall mean and standard deviation from the generic `scipy.stats.norm.fit()` fit as starting parameters for the compound models.
 - The algorithm iterates through trial compound fits of increasing amplitude, accepting every fit that meet loose requirements for the returned standard deviation and amplitude. These fits help to calculate a more precise mean and standard deviation, which are used as the input parameters for the compound models in the next step.

2. Strict Requirement Iterative Compound Fitting:

- Using the mean and standard deviation parameters from the previous step as more precise model input parameters, this phase generates new compound models and again, iterates through various increasing amplitude values.
- The algorithm employs much more strict requirements on the returned fitting values, ensuring the returned parameters (mean, standard deviation, amplitude) fall within acceptable ranges that ensure the fit was a success.
- In addition to the returned fitting parameter requirements, a synthetic dataset is also generated to produce a histogram that similarly follows the shape of compound model. The stricter requirements also require a `ks_statistic` and an earth mover's distance `EMD_metric` to fall within acceptable ranges that guarantee the model's goodness-of-fit.
- Parameters from successful fits are stored for further analysis.

1.3.3 Error Calculation

If a compound model passes the strict requirements, the error of the mean is calculated using one of two methods, allowing the user to choose.

1. Bootstrapping:

This method is computationally intensive and involves repeatedly resampling the data and fitting the compound model to histograms of each resampled dataset. The variance of the mean is estimated from the standard deviation of the returned mean parameters of every fit.

2. Weighted Gaussian and Linear Error Calculation:

For the Gaussian part of the compound mode, the error is calculated as

$$\varepsilon_{\text{Gaussian}} = \frac{\sigma_{\text{Gaussian}}}{\sqrt{N_{\text{Gaussian}}}}$$

where N_{Gaussian} is the number of stars within three standard deviations (σ_{Gaussian}) of the returned mean.

For the Linear part, the error is derived from the standard deviation of histogram bin residuals for bins outside the Gaussian region. The linear error is calculated as

$$\varepsilon_{\text{Linear}} = \frac{\sigma_{\text{Linear}}}{\sqrt{N_{\text{Linear}}}}$$

where N_{Linear} is the number of stars under the linear-part of the fit.

These errors are then weighted by the ratio of stars under each part of the model:

$$\varepsilon_{\text{overall}} = \sqrt{\text{gaussian_weight} \cdot \varepsilon_{\text{Gaussian}}^2 + \text{linear_weight} \cdot \varepsilon_{\text{Linear}}^2}$$

This error calculation is repeated for histogram bin numbers in the range of $\text{int}(\sqrt{N_{\text{total}}})$ to $\text{int}(\sqrt{N_{\text{total}} + 10})$. The minimum error from these calculations is chosen as the overall error.

I have found this error almost nearly matches the error derived via bootstrapping for most cases. This runs much faster however.

The entire OCF algorithm is repeated for initial `catalogy` histogram bin numbers ranging from eight to twenty. This is to ensure the optimal successful bin parameters are found, as increasing the number of histogram bins affects the generated compound models.

After completing all the compound fitting iterations, the algorithm selects the parameters of the fit that correspond to the minimum error. These optimized parameters provide an accurate and reliable mean magnitude for the starting bin.

1.3.4 Handling OCF Algorithm Failures

There is a possibility that the OCF algorithm might fail to find any successful compound models, even after iterating through various amplitude values. The primary reason for this failure is often the lack of sufficient stars in the bin to fit an effective `Gaussian1D()` + `Linear1D()` compound model.

To address this issue, the algorithm includes a mechanism to extend the bin in color-space and reattempt the fitting process:

1. Initial Failure Handling:

- If the OCF Algorithm fails for the starting bin, it extends the bin by 0.1 in color-space of the CMD to include more stars.
- The algorithm then again extracts the stars in the new, larger bin and re-runs the OCF algorithm on the extended bin.

2. Iterative Expansion:

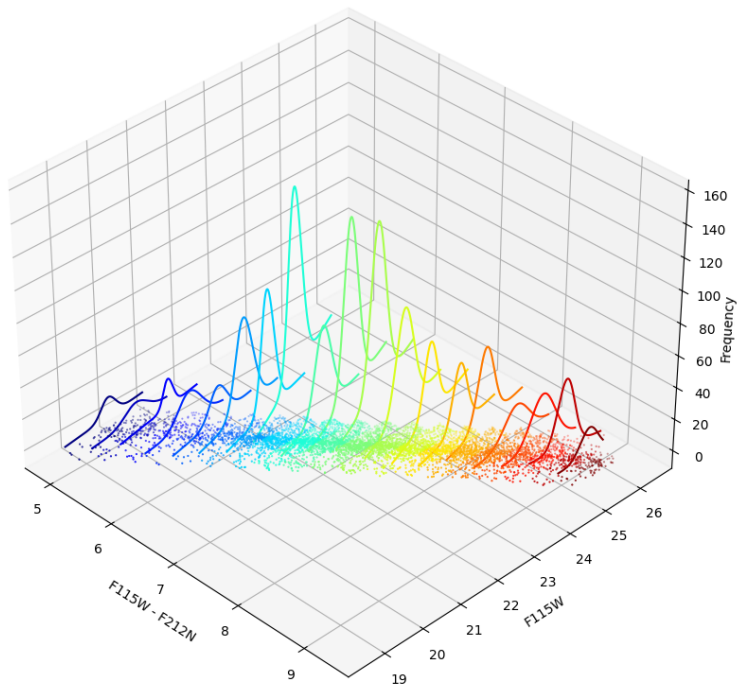
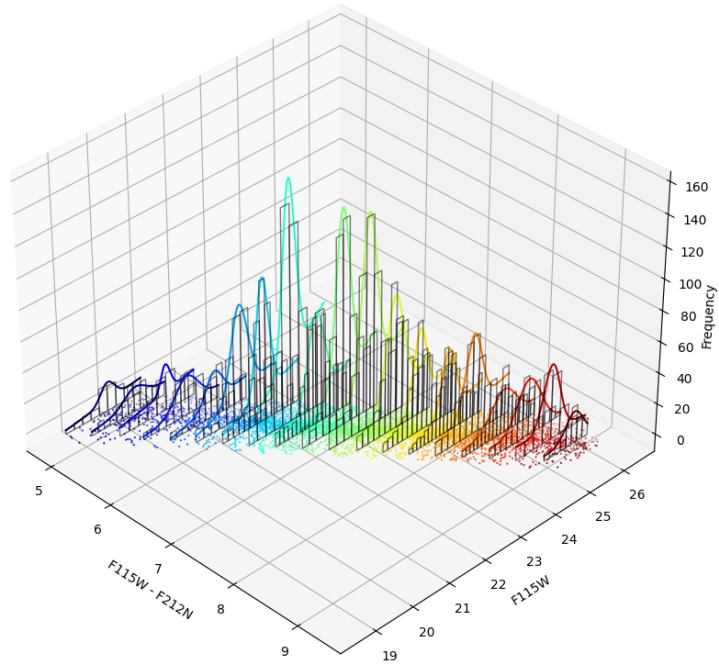
- If the OCF algorithm fails again, the bin is extended another 0.1 in color-space.
- This process is repeated up to 7 times, each time extending the bin by 0.1, to include a progressively larger number of stars.

3. Final Failure Condition:

- If the OCF algorithm to find a successful compound model after 7 extensions (a total extension of 0.7), it indicates a deeper issue.
- In this case, the failure is likely due to improper initial cutoffs (x -range / parallel line parameters). From my experience it is mainly from the parallel line parameters, after the 3x width increase, extending into the main-sequence part of the CMD, which prevents a single one-peak compound model from succeeding.

After completing the OCF algorithm for the starting bin, the algorithm proceeds by generating a new bin starting at the end of the previous bin. This new bin similarly begins having a width of `x_range / n`, and OCF is again implemented on the new bin. This iterative process continues, with each new bin starting where the previous one ends, until the rightmost x value of a bin exceeds the end of the provided x -range. OCF is applied one last time to this final bin and the algorithm concludes.

After a successful OCF run across the RC cluster, the algorithm outputs a plot like the one below for the example CMD:



The slope is finally calculated using a weighted `scipy.optimize.curve_fit()` linear fit which produces the following plot in Figure 3:

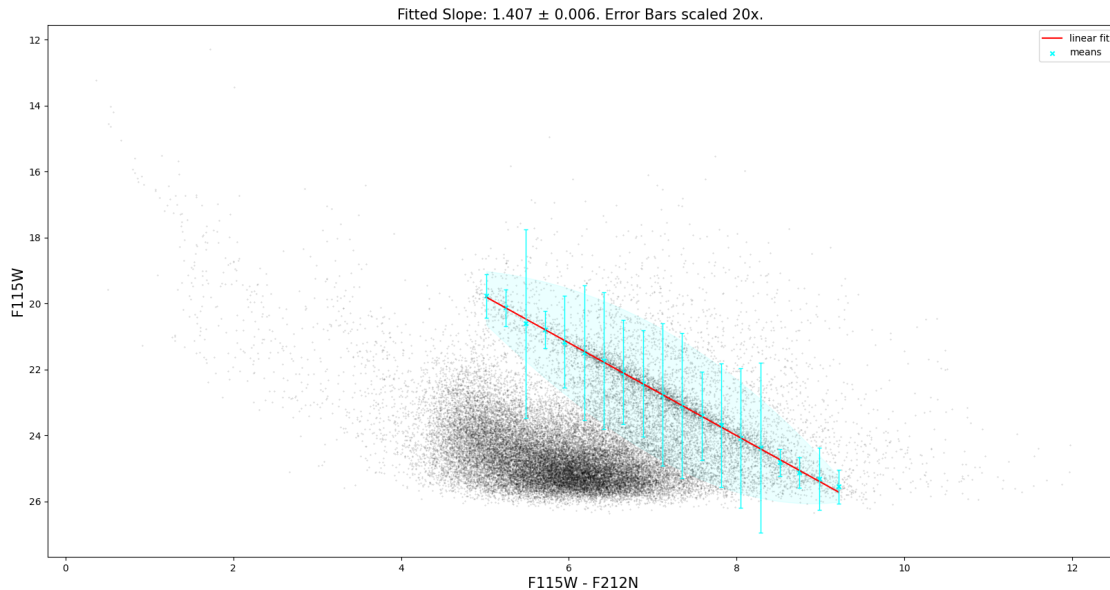


Figure 3: Calculated Slope using OCF algorithm for example CMD. Note that the Error Bars are scaled by 20x.

1.4 Goodness-of-Fit Assesment for Entire Compound Model Fitting

To ensure the accuracy and reliability of the derived RC (Red Clump) slope from the multi-compound model fitting, we assess the goodness-of-fit by generating a synthetic RC cluster using the optimized compound models and performing subsequent statistical tests.

1. Generating Synthetic RC Clusters:

- For each tile of the RC cluster, a synthetic RC cluster is generating using its corresponding optimized compound model.
- This synthetic cluster is designed to mimic the distribution of the actual RC tile. Example synthetic RC Tile clusters are shown in Figure 4.

2. Performing the Kolmogorov-Smirnov (KS) Test:

- The KS test is employed to statistically compare the entire collection of synthetic RC clusters against the actual RC cluster.
- This test evaluates the goodness-of-fit by calculating the distance (`ks_statistic`) between the empirical cumulative distribution functions (CDFs) of the synthetic and actual RC clusters.
- A good fit, based on the KS test, thus has a `ks_statistic` close to 0.

3. Plotting CDFs and Density Plots:

- To visually inspect the similarity between the synthetic and actual RC clusters, CDFs and density plots are generated.
- These plots help in identifying any significant deviations between the distributions of the synthetic and actual RC clusters.
- Example CDF and Density Plots are shown in Figure 5 and 6, respectively.

When plotted using identical x -values, the synthetic cluster appears visually distinct from the actual RC cluster. This difference arises because each tile is generated from a single optimized compound model, resulting in a synthetic cluster that only follows the characteristics (histogram) of that model rather than the exact visual traits of the actual RC cluster. But for the purposes of calculating the mean, the histograms closely resembling each other is what matters.

After the goodness-of-fit has been established, several statistical plots can be generated:

1.4.1 Plotting Synthetic vs. Actual RC clusters on Example CMD

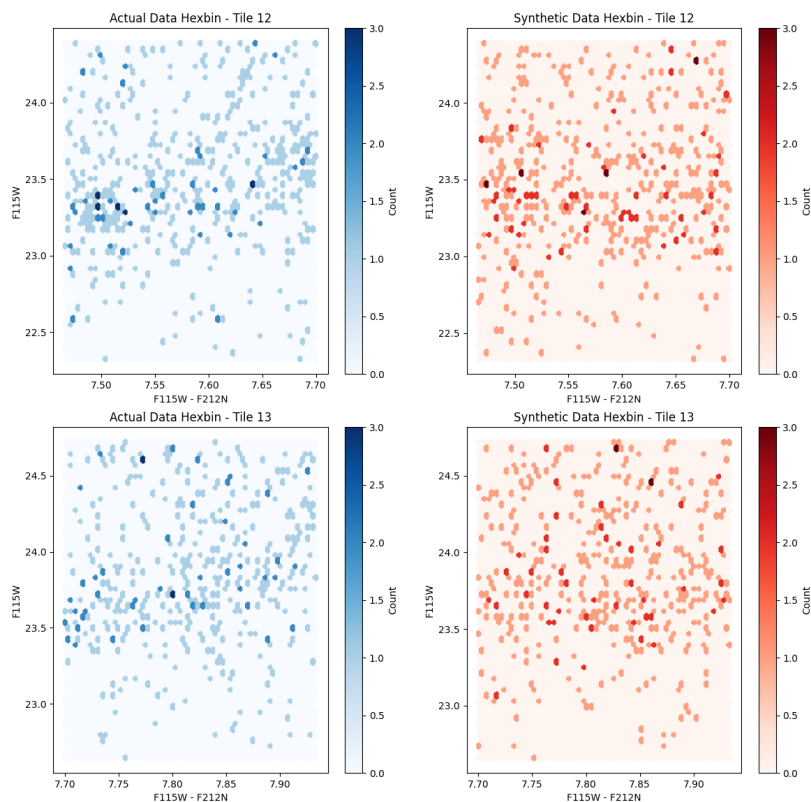


Figure 4: Hexbin plots comparing synthetic vs. actual RC tile data for tiles 12 and 13 from an $n = 18$ run.

1.4.2 Plotting CDFs

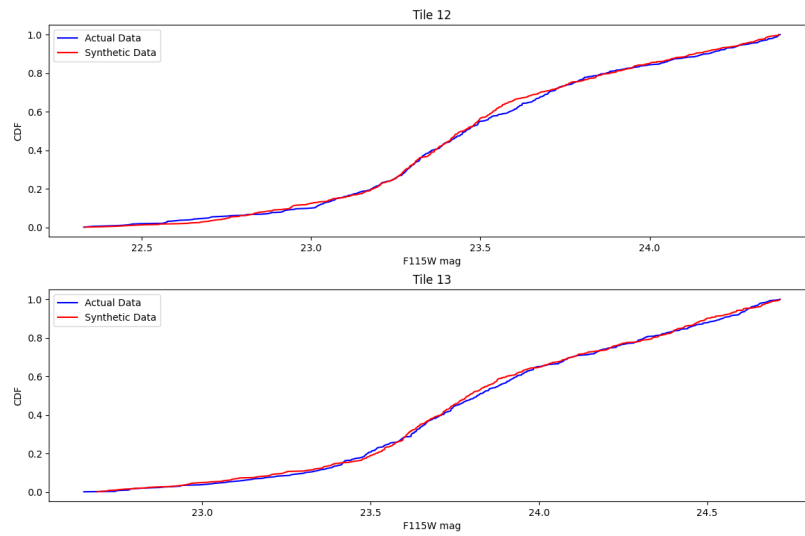


Figure 5: Overlaid synthetic & actual CDFs for tiles 12 and 13 from an $n = 18$ run.

1.4.3 Plotting Densities

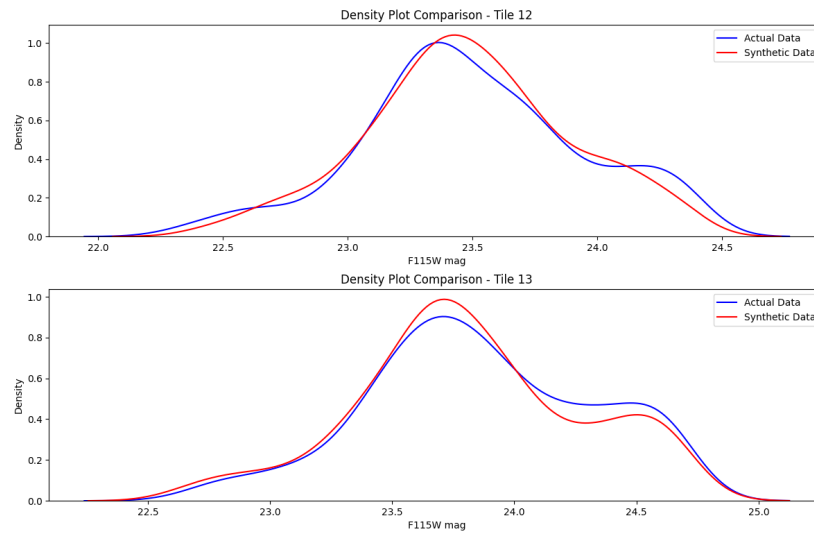


Figure 6: Overlaid density plots generated using a KDE for tiles 12 and 13 from an $n = 18$ run.

And finally, plotting the residuals which as a histogram, should resemble a Gaussian.

1.4.4 Plotting Residuals

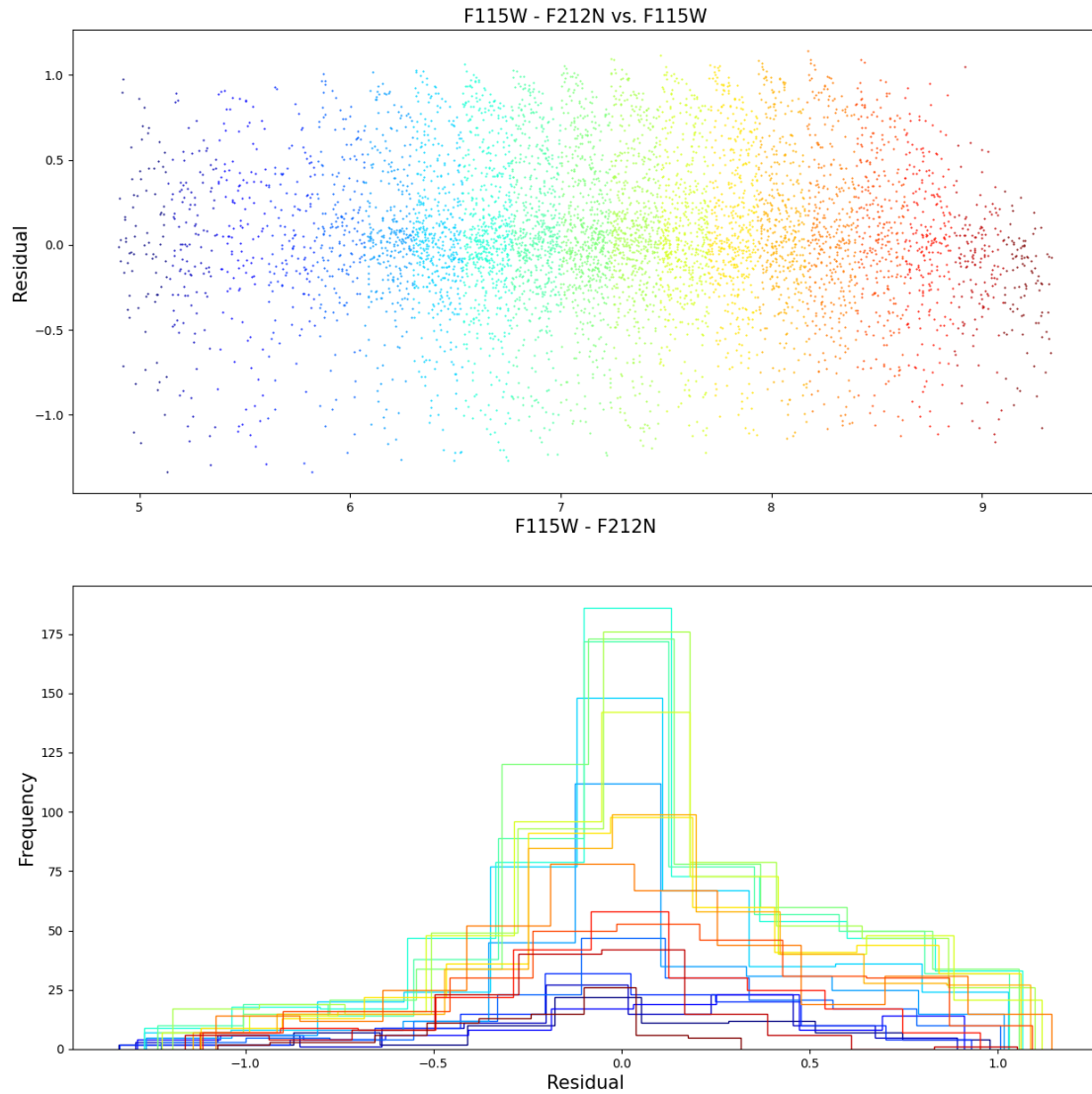


Figure 7: Residuals from the calculated linear fit from the OCF algorithm from an $n = 18$ run.

1.5 Running the Algorithm

The algorithm requires an input parameter n , an integer telling the algorithm how many tiles to segment the RC bar into. However, different values of n could result in calculating different slopes.

For this reason, the algorithm lets a user input an array of different n s they wish to run the

algorithm for. It then finds the optimal n such that the *slope*-error is minimized and generates the above plots for the optimal n value.

It also generates a plot like below, allowing one to compare the results of different n and any patterns that emerge across calculated slopes and errors as n increases.

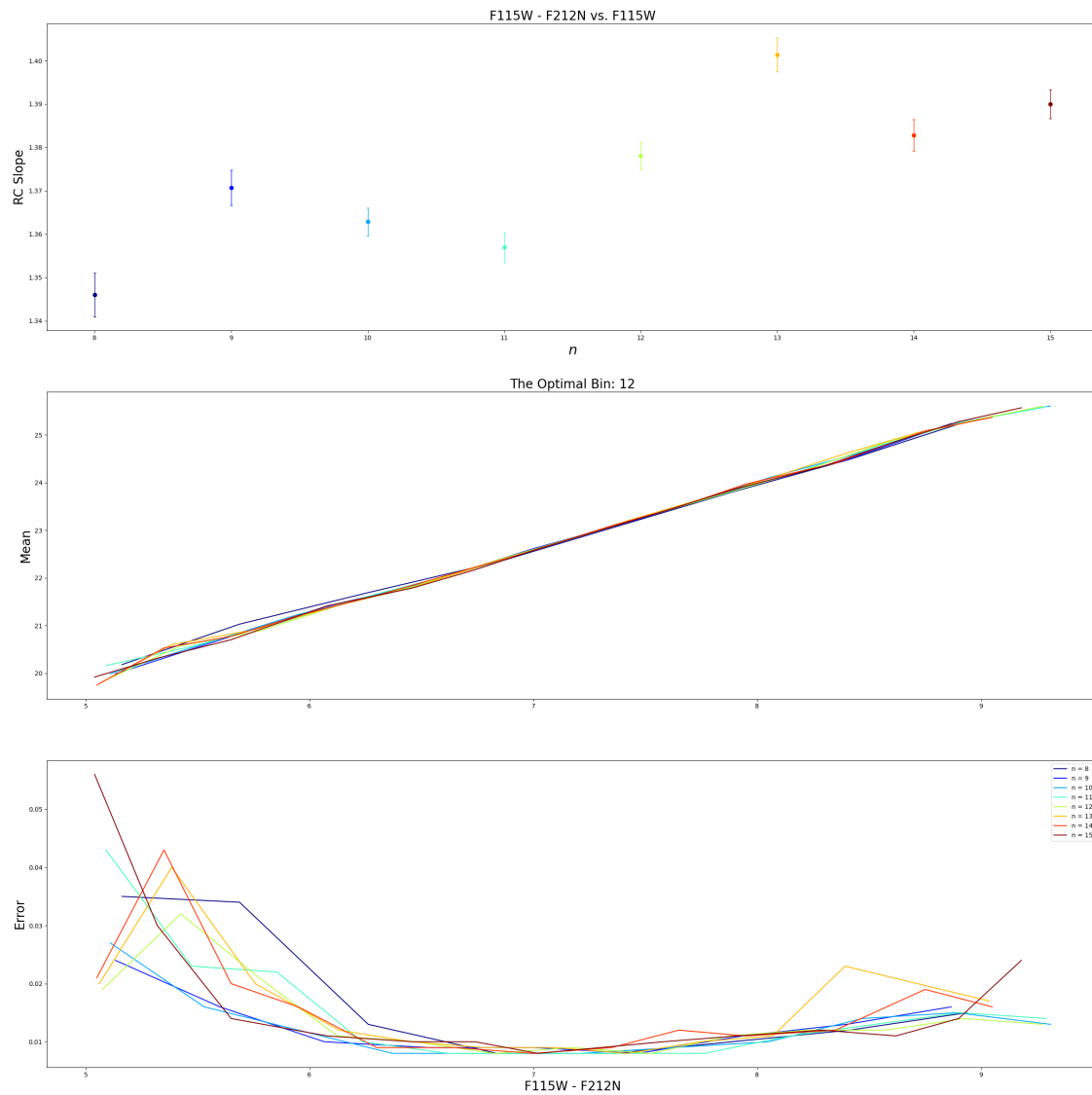


Figure 8: Comparing OCF slope and error results across different n

2 Alternative OCF Algorithm – Rectangles

For CMDs that look like this:

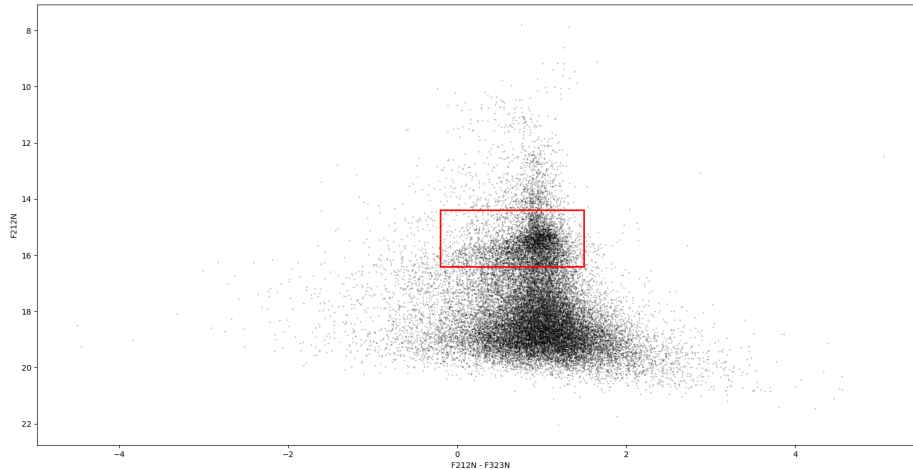


Figure 9: NRCB1 F212N - F323N vs. F212 CMD


where it is potentially difficult to confidently generate the required parallel line RC cutoff input parameters for the tiled-OCF algorithm detailed above, there is another option using an overall rectangular cutoff rather than two parallel cutoffs.

This is fairly self-explanatory. Rather than generating tiled-bins that follow the parallel lines across the slope of the RC, the algorithm generates equal-height tiles that span the RC cluster. This method replaces the parallel cutoff input parameters with an `xlim` and `yylim` parameter that defines the overall rectangular RC cutoff within which the tiles are generated.

The rest of the OCF algorithm is identical.

Note that this method is discouraged as the main method of calculating RC slopes. I recommend using this method only to calculate a slope from which the parallel cutoffs of the first method can better be determined. Usually though, unsharp-masking is more than good enough.

In addition, more often than not, it is impossible to generate a perfect rectangle RC cutoff using an `xlim` and a `yylim` that does not cross into the main-sequence portion of the CMD (which would fail the OCF algorithm).

 – `red_clump_riemann.py`, `red_clump_script.py`.