Dynamic Aperture and Dynamic Binning

Alexander Heckett

August 25, 2015

Abstract

Traditional, astronomical, photometric methods of computing star intensities and calculating uncertainties in measurements lead to unnecessarily large error bars. Dynamic Aperture and Dynamic Binning are clever ways of reducing error both in the computation of star intensities and in their analysis that allow for smaller fluctuations in star brightness to be detected. These methods are explored conceptually, grounded algorithmically, and implemented with dramatic results. When used in sync, these two methods frequently halve error bars.

Contents

1 Background

1.1 Summary

Over the last week, I have been working on a program that could significantly aid in the detection of fluctuations of astronomical objects, such as the identification of exoplanet transits, quasar and black hole monitoring, or any other varying electromagnetic source. The results are pleasingly dramatic, while the computation time required to attain them is only a few seconds on even quite slow computers. Furthermore, they are currently implemented in Python, which is a programming language known for ease of programming but a slow runtime, further illustrating the lack of computation power required to run these procedures.

1.2 Terminology

Before lunging into the details of the project, a quick review of background topics is appropriate. Lightcurves are simple plots of the emissions of an electromagnetic source over time. The number of photons (or any other unit) is recorded on the y-axis, while the time (equivalently, frame number) is marked on the x-axis. Before we can conclude anything about an object's behavior, we must know something about how certain our measurements are. For instance, there is no detectable difference between an intensity of 100,000 and 100,050 photons if we are only certain of our measurements to within 10,000. However, calculating an uncertainty can be very difficult, so it is often preferred to use the dataset itself to deduce this. We partition up our dataset into chunks within which we believe our object does not vary. We then conclude that all fluctuation within such a chunk (formally known as a *bin* or a *block*) must be noise due to our detection mechanism. We assume that any one of those measurements must have been somewhere between the minimum and the maximum of that bin, but can say no more about them with much confidence. The difference between the maximum and the minimum is known as the height of the *error bar*.

Inherent in this procedure is a fundamental tradeoff: accuracy versus confidence. We can choose to have very large bins, in which case we are virtually *certain* a measurement will lie between the minimum and the maximum, but because we have a large number of frames in our block, we lose the fine detail of the structure of the lightcurve. On the complete contrary, we can chose to have tiny bins, in which case all fine detail is preserved, but we have little faith that a measurement will lie between the minimum and maximum, because we have only shown this to be true for a small number of frames. Picking what bin size in-between these two opposites to go with is one task that drastically effects what changes in intensity are detected by a program.

2 Concepts

2.1 Existing Methods

At this point, we are prepared to hear the traditional route taken for computing the intensity at any moment of a source in the sky, as well as the partitioning up of the lightcurve of such intensities into bins. The intensity of a star (or any other object) is defined to be the sum over all pixels inside a circle centered somewhere in the source that completely encompasses the star, as well as a good deal of the space around that object. Optionally, at very little computational cost, this can be made more accurate by taking some sample of pure space (no additional stars in this region) near our star as representative of the local background noise, whose average brightness per pixel can be taken off the sum previously found. This method does indeed sum over the entire star, but it also sums over a good deal of empty space. I shall call this *Static Aperture*, because the aperture of the star (the region of space we sum over) does not change from frame to frame.

The most common method for finding the partitioning to use is to take blocks of a fixed size. This rigidly defines the appropriate place to put the tradeoff between accuracy and confidence, ignoring any opportunities to, say, move the boundaries of these blocks around by just a few frames in order to dodge large spikes, or even to optimize the heights of these blocks depending on the local shape of the lightcurve. I will call this *Static Binning*, because the bins are all of the same size.

2.2 New Ideas

At this stage, we are in a position to make some cuts to the error bar heights, allowing finer detail to be seen in these lightcurves. Let's start by changing Static Aperture to a new method for computing intensity, *Dynamic Aperture*. When computing intensities via Dynamic Aperture, at every frame we calculate the exact boundary of the star, pixel by pixel, and sum over that region, hence the name "Dynamic." Since we are summing over fewer pixels, we are introducing less uncertainty, and hence reducing the error of our measurements.

Another improvement we can make is to use *Dynamic Binning*, as opposed to Static Binning, in which we customize our bins to have different sizes from each other. This allows for the program to make optimizations that would otherwise be impossible, hence reducing the observed error bars, even if we are not improving the lightcurve itself.

3 Algorithms

3.1 Dynamic Aperture

Now, we must figure out a way to execute our plan. First, we need to attempt to implement a Dynamic Aperture. We wish to separate the bright star from the relatively dim background, so a good starting point would be to make this distinction precise. We dip into

our pool of "pure background" to find the mean and standard deviation of the background pixels. Assuming that the brightness of background pixels follows a Gaussian distribution, we can use this to figure out what cutoff, say, $\frac{1}{50}$ of the background is above. Once we have this cutoff, we go look inside our circle that the star must be completely contained in to figure out what pixels satisfy this requirement. The star can be assumed to be so bright that all pixels containing the star appear. As well, many unwanted, stray bright pixels will be included. The task at this point is to separate the star from the strays. While it is highly probable that *some* background pixels will be included, it is *extremely* improbable that they will all be connected. However, the star definitely will be in one piece. Therefore, all we have to do to find the star is to detect the largest contiguous block of extraordinarily bright pixels. This can be done by taking a pixel in our set of bright specks, exploring up, right, down, and left until we have traced out the patch of other bright specks it is connected to, and seeing if we have already found another patch with this many connected. If not, then set this to be our record. Of course, if a bright speck has already been visited, don't bother re-exploring that same patch again. At the end of this procedure, we will have the largest contiguous block of bright pixels together. This is how we define our star.

3.2 Dynamic Binning

Our task for executing Dynamic Binning is no easier. The first, and by far hardest, part of this task is to define what the word *better* means. I define a better partitioning of a lightcurve to mean that we get more information about it, which in turn I define to mean a lower average standard deviation, weighted by the number of frames each bin has. However, since I only have error bar heights to work with, there is no point in talking about the standard deviation of the data directly. We can only infer the standard deviation of a bin given how tall the bar is and how many frames the block contains. How do we make this inference? First, we pre-compute the average heights of bars with a certain number of frames in them, given that the data we have has mean 0 and standard deviation 1, and save this as a .txt file for many different numbers of frames (in my case, I pre-computed 2 to 500). I'll call this function $\zeta(n)$, where n is the number of frames we have in a bin. Then, we can use this .txt file to draw the conclusion that:

$$\sigma = \frac{h}{\zeta(n)} \tag{1}$$

Where h is the height of our error bar, σ is the standard deviation of the data within our bin, and n is the size of our sample. Then plugging this formula into an average gives a metric for the "badness" of a collection of bins (the quantity which we will ask our program to minimize):

$$b \equiv \frac{\sum_i \left[\frac{nh_i}{\zeta(n_i)}\right]}{\sum_i [n_i]} \tag{2}$$

Since $\sum_i [n_i]$ is just the total number of frames we're analyzing, it doesn't make sense to divide by this constant, since b can be multiplied by any (positive) number to give an equally valid "badness metric." This lets us define $\tilde{b}$:

$$\tilde{b} \equiv \sum_i \left[\frac{n h_i}{\zeta(n_i)} \right] \tag{3}$$

We have here a curious phenomenon. The badness of this collection of bins is a sum of "badnesses" for each bin:

$$\tilde{b}_i \equiv \frac{n h_i}{\zeta(n_i)} \tag{4}$$

This convenience allows us to write an extremely speedy algorithm for finding the very best partitioning to use, as opposed to needing to check every different possibility, which would make this problem completely impossible computationally! We can handle this problem recursively. Let's say we know the best ways to partition the lightcurve up, starting at every place to the right of some given position. We also know the total "badness" of partitioning the lightcurve from that point on that way. Well, to find the best way to partition the lightcurve up from our given position, we just check the badness of the partitioning from the test location on, plus the little additional badness of the block we would need from our given "root" to the test location, compared to the best known solution (a long block stretching from our root to the very end of the lightcurve). Therefore, n items until the end of the lightcurve, it takes $O(n)$ additional pieces of computation for every unit you go to the left. That makes this algorithm $O(n^2)$. Compared to an exponential, $O(n^2)$ looks pretty good!

In fact, we're not even done. Our algorithm currently finds the best possible partitioning. However, we can place constraints on this program that both speed it up (by requiring it to look over fewer possibilities) and place bounds on the types of partitionings it can output. Firstly, we can set an upper limit on how large a bin we will accept. If we are expecting fluctuations on the order of, say, 500 frames, we can accept bins no larger than, perhaps, 750 frames, because blocks that are longer will run the risk of hiding important changes. The main purpose of this limit, however, is really to speed up the program. Since we only need to look for junctions a fixed distance forward, our $O(n^2)$ algorithm becomes $O(n)$! Granted, a better way of writing this would probably be $O(nm)$, where m is the maximum number of frames a block can contain. This n dependence is the best that could possibly be conceived. For a quick thought-experiment style proof, we could make some frame arbitrarily large, which can make a given partitioning arbitrarily unprofitable compared to one in which that frame is in a bin of size 2 (if we have an odd length lightcurve, not every bin can have size 2, thus there is at least one location that, if we "inflate" its value, can be shown to be in a sub-optimal bin). Thus, we have to at least look at every value in the lightcurve, which means that any algorithm attempting to find the best partitioning must be at least $O(n)$.

We can equally well set a lower limit on the number of frames in a bin. This gives us an upper bound on the probability that a measurement will leak outside our error bars, which

is given by the formula:

$$p = \frac{2}{n+1} \tag{5}$$

Thus, we have two algorithms, Dynamic Aperture and Dynamic Binning, which reduce our uncertainty in star brightness measurements.

4 Results

4.1 Statistics

I have implemented these two methods in Python. Now we get to examine their results. For my dataset (1,712 frames of PG 1718+481), I have taken a Dynamic Aperture P-value of 0.05, a Dynamic Binning unspecified maximum length, and a Dynamic Binning maximum P-value of 0.2. The average length of a bin, weighted by the number of frames in each bin, is 37.85. This indicates that the program often found it profitable to make larger bins with higher sizes to allow for larger ζ. The smallest bin my program generated is of size 10, meaning that at least once my program maxed out the limit on bin sizes (on the lower end, at least). One (weighted) quarter of my bin sizes are at or below 15, while the median bin size is 23 (again, weighted by the number of frames a bin contains). Since $23 < 37.85$, we have reason to believe that the distribution of weighted bin sizes is skewed to the right. Three quarters of the weighted bin sizes are below 43, while the largest bin size is 132. If we set the static block size to 38 for a side-by-side comparison of performance, we see that using Dynamic Blocking with the same average (weighted) bin size as Static Blocking reduces the average error bar heights by a factor of 1.255. It also reduces $\tilde{b}$ by a factor of 1.158. Using Dynamic Aperture further reduces the weighted average of error bar heights by a factor of 1.554, coming out to a grand total of a 1.951 factor reduction! This large reduction factor has been purchased at only a couple of seconds of computation time, making Dynamic Aperture in junction with Dynamic Blocking an effective way of reducing the uncertainty in star intensity measurements.

4.2 Figures

As a second form of result, we can examine some plots and visual displays of our program. To demonstrate Dynamic Aperture visually, Figure 1 is a sample "star" detected by my code. Everything inside the blue border is the star. The border itself consists of the closest pixels to the star that aren't included as part of the source. Judging by eye, it looks as though our algorithm is doing a good job! For Figure 2, I have taken my test dataset and plotted a lightcurve of one star in the photograph. On the left of the purple dividing line, I have both used Dynamic Aperture to reduce the fluctuation of the lightcurve and Dynamic Binning to get the best bound on the data. On the right, I have used neither. Clearly, these techniques

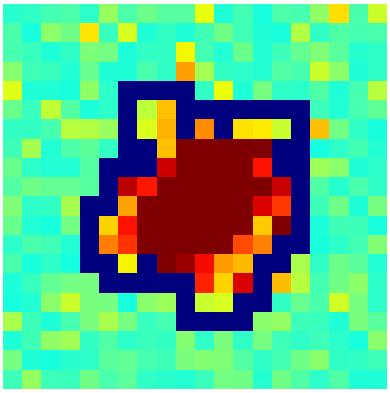

Figure 1: A blown up image of a star, with pixels closer to red representing more light, and pixels closer to blue representing less. The blue ring is the border of the star as found by the Dynamic Aperture program.

reduce the size of the error bars! We can also demonstrate the reduction in error bar height by plotting them as two different curves on the same graph, as is done in Figures 3 through 6 for different sizes of the Static Bins.

5 Conclusion

These statistics and graphs all show the merit of using Dynamic Aperture and Dynamic Bins. The performance of these algorithms rests on a good choice of their parameters. For Dynamic Aperture, there is one parameter to tweak: the P-value. Too small a P-value can cut out the dimmest pixels in a star, making our intensities too low. Meanwhile, the error bars will be fantastically small. Too large a P-value will accurately sum up *all* of the star with complete certainty, but may include more pixels as a result and hence a larger error bar. Dynamic Binning introduces two more variables: the maximum acceptable P-value and the maximum bin size. The maximum bin size is mostly a convenience of speeding up the program - the bins tend to stay closer to the lower limit induced by the maximum P-value of a measurement lying outside the error bar than by a maximum number of frames. This second parameter has a substantial impact on the output, quantitatively dictating how much advantage should be taken of small regions with low errors. Reasonable values of it are generally somewhere between 0.05 and 0.2, though of course circumstance should have the final say on how tolerant of mistakes we ought to be. All in all, though, irrespective of how immaculate a job we do of choosing our parameters fastidiously, Dynamic Aperture and Dynamic Bins allow for lower uncertainties in lightcurves, finer detail in fluctuations, and a better chance of detecting changes in source brightness, all at very little computational cost.

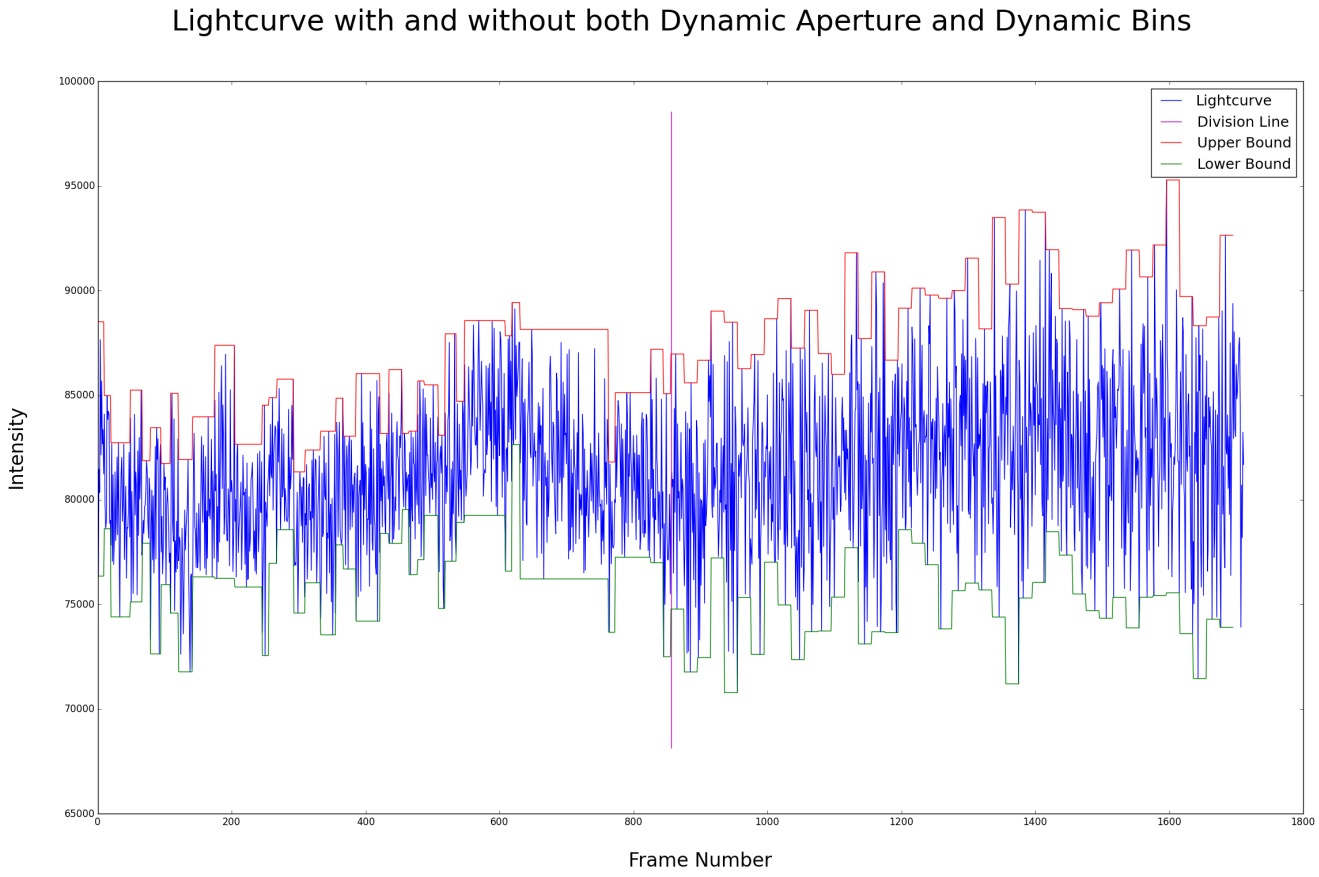

Figure 2: The lightcurve of a star throughout 1712 consecutive frames. The left half of the image has been processed by both Dynamic Aperture and Dynamic Binning, while the right half uses conventional methods.

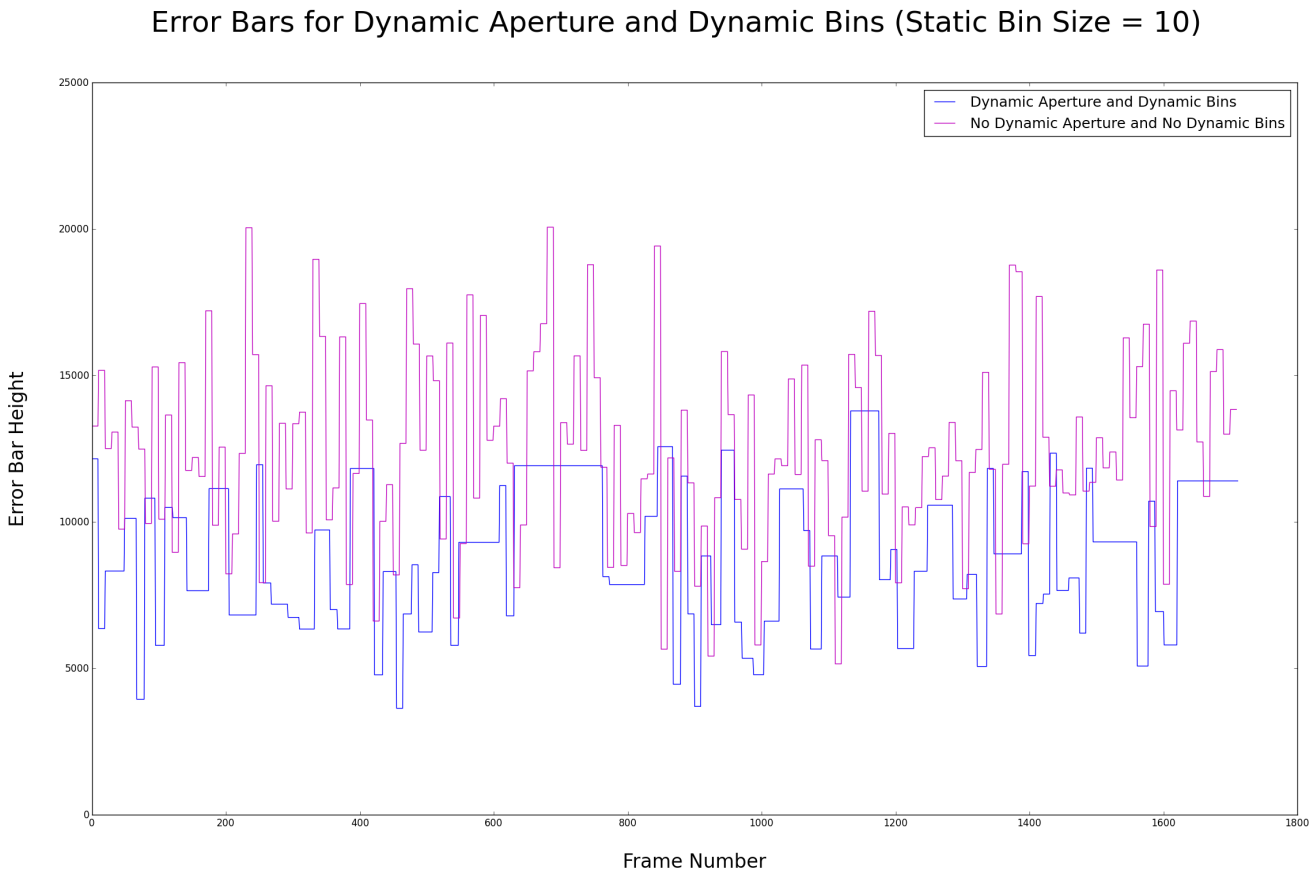

Figure 3: Dynamic Aperture and Dynamic Blocking alongside neither. The Static Bin Size is 10.

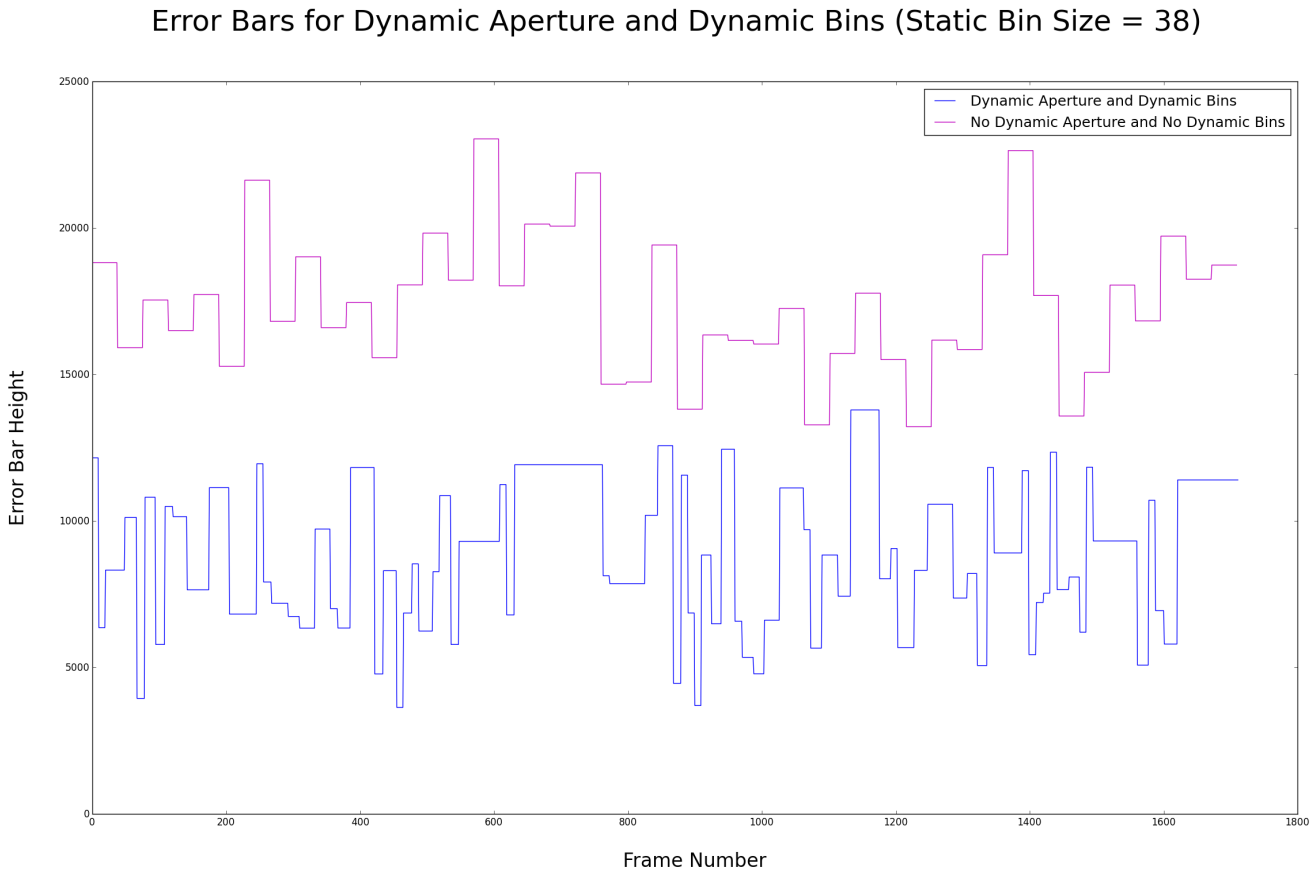

Figure 4: Dynamic Aperture and Dynamic Blocking alongside neither. The Static Bin Size is 38, which is also the weighted average bin size of Dynamic Aperture. Thus, we have a fair comparison.

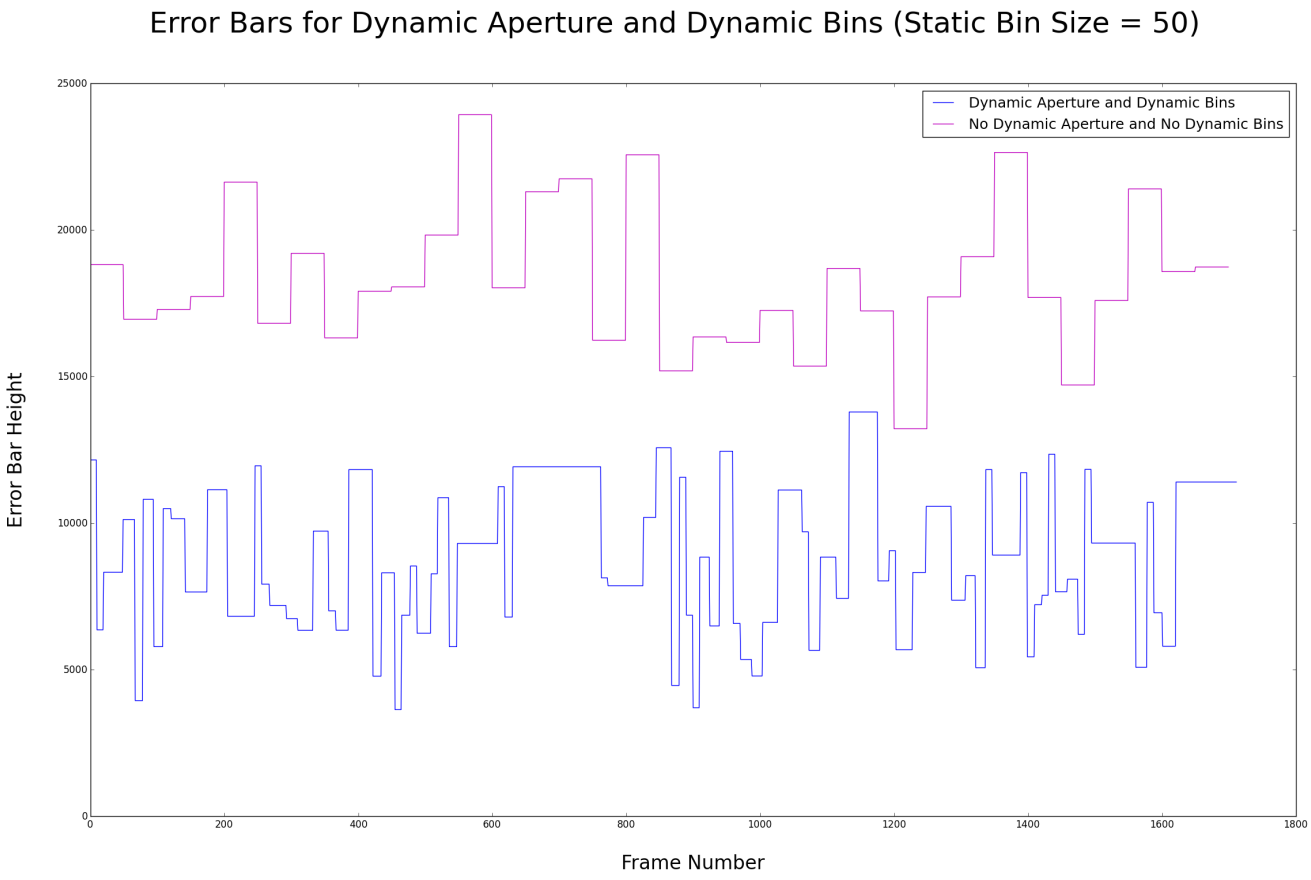

Figure 5: Dynamic Aperture and Dynamic Blocking alongside neither. The Static Bin Size is 50.

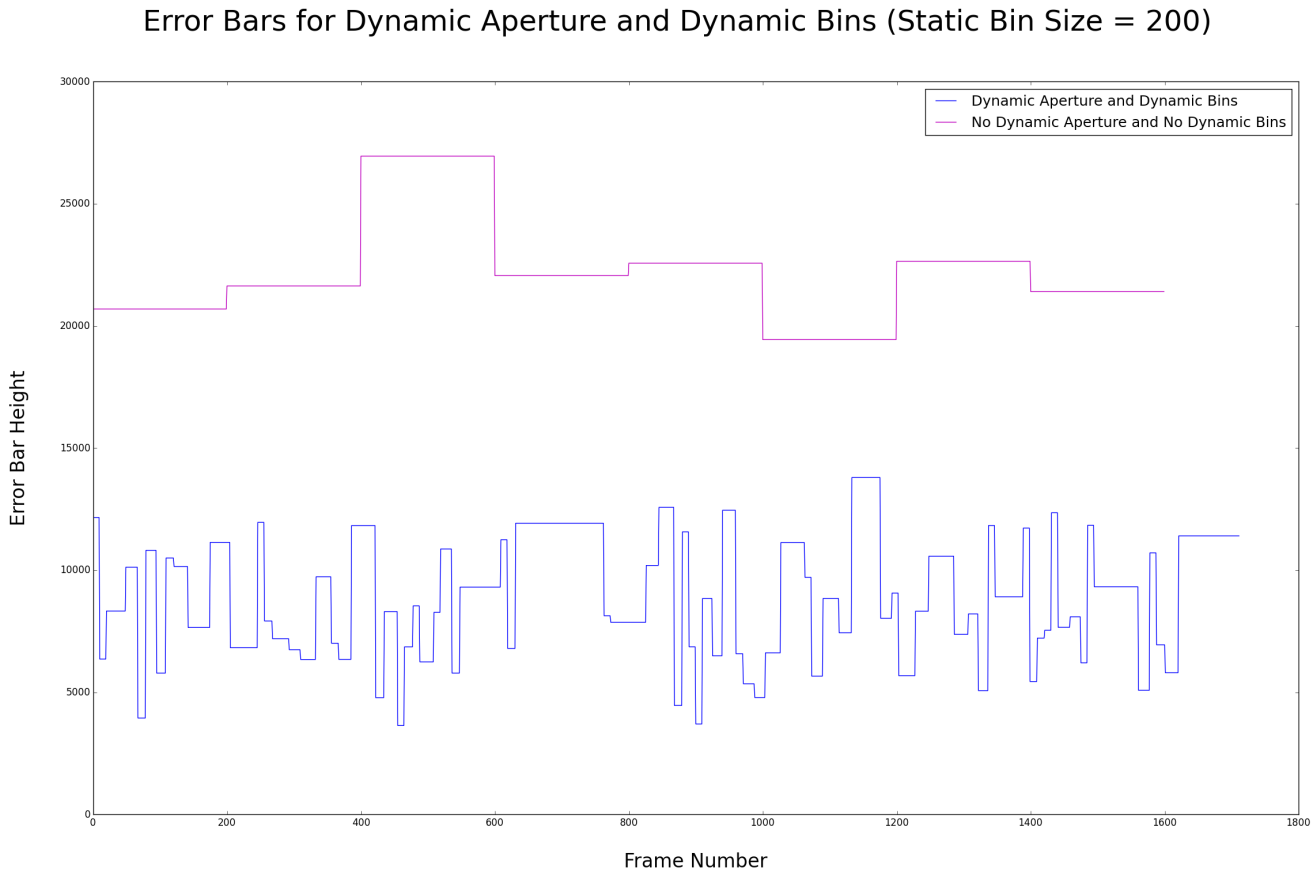

Figure 6: Dynamic Aperture and Dynamic Blocking alongside neither. The Static Bin Size is 200.