Mode with clustered averages

Bradley Thomas

Code

```
keep_mode=np.array([])
save=np.array([])
averages=np.array([])

#use the averaging method defined above and then get mode

#first we do rows
for a in range(len(img2)):
    valmax=np.amax(img2[a]) #get the actual max value in the array
    lowerbound=.85*valmax #we want to see where values above this are. The high intensity values. I chose a 15% threshold.
    for i in range(len(img2[a])): #go through one row at a time
        if img2[a][i]>lowerbound:
            save=np.append(save,i) #get indices of higher values in an array

    averages=np.append(averages,cluster_avg(save))
    save=np.array([])

vals,counts = np.unique(averages, return_counts=True)
mode = np.argmax(counts)
true_mode=vals[mode]


keep_mode=np.append(keep_mode, true_mode)

print("Approximate horizontal (row) center: "+str(true_mode)) #nanmean ignores nan values. nan means not a number. If we just do
```

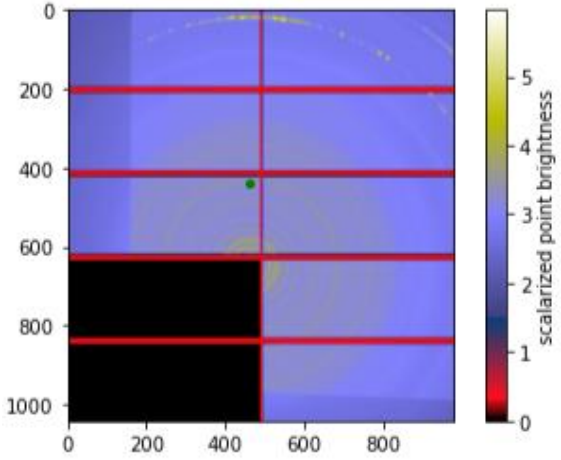
```
#for columns
averages=np.array([])
save=np.array([])
for b in range(len(img2[0])): #go through every column and note that img2[0] is the size of the number of columns.
    valmax=np.amax(img2[:,b]) #get the actual max value in that column.
    lowerbound=.85*valmax #we want to see where values above this are. The high intensity values. I chose a 15% threshold.
    for j in range (len(img2[:,b])): #go through each index of the column
        if img2[:,b][j]>lowerbound: #if the element is larger than our bound
            save=np.append(save,j) #get the indicies of the larger elements and add them all to a list

    averages=np.append(averages,cluster_avg(save)) #the average of the larger elements (b) to find the center of high intensity s


vals,counts = np.unique(averages, return_counts=True)
mode = np.argmax(counts)
true_mode=vals[mode]


keep_mode=np.append(keep_mode, true_mode)

print("Approximate vertical (column) center: "+str(true_mode)) #nanmean ignores nan values. nan means not a number. If we just d



print("["+str(keep_mode[0])+","+str(keep_mode[1])+"]")
```

```
def cluster_avg(save):
    hold_avgs=np.array([])
    hold_indices=np.array([])
    lastindex=0
    for j in range((len(save)-1)):#We don't want to go out of bounds. so keep the j+1 from going too far.
        if(save[j+1]>(save[j]+10)): #The +10 is just a number I have to be the amount above to split. So if values differ by 10,
            for k in range(lastindex, (j+1)): #go from previous cluster's last index to the next one that starts a new cluster.
                hold_indices=np.append(hold_indices, save[k]) #put the cluster's indices in an array
            hold_avgs=np.append(hold_avgs, np.nanmean(hold_indices)) #put the mean of the cluster's indices in an array
            hold_indices=np.array([]) #empty the array that holds the indices
            lastindex=j+1 # move the last index up so we will not look at the same cluster again in our for loop

        elif(save[j+1]==save[-1]):
            for k in range(lastindex, (j+1)): #go from previous cluster's last index to the next one that starts a new cluster.
                hold_indices=np.append(hold_indices, save[k]) #put the cluster's indices in an array
            hold_indices=np.append(hold_indices, save[-1]) #include the final element of save
            hold_avgs=np.append(hold_avgs, np.nanmean(hold_indices)) #put the mean of the cluster's indices in an array
            hold_indices=np.array([]) #empty the array that holds the indices
            lastindex=j+1 # move the last index up so we will not look at the same cluster again in our for loop
    avg_of_clusters=np.nanmean(hold_avgs)

    return(avg_of_clusters)
```

- I decided to not use scipy. Instead, I created an algorithm to do as follows:
- We can have a for loop going through an array called save. Save contains all indices for values in a row or column above our threshold of 0.85 the max value in that same row or column.
- When save[j+1]>(save[j]+10) we can loop from the beginning to that index and add up the values. Then we can average them and put that average into an array. We can go to where we left off in the array. Then we can continue forward until save[j+1]>(save[j]+10) is true again. We can add up the values from after the last index that was part of the previous average calculation. Then we average these and they go into an array. We go back to where we left off, and this repeats until we get to the end. When we reach the end of save, we do the averaging from the last index we looped to, to the final element (so we don't ignore the final section of our array).
- At the end we average (using nanmean) the array holding the averages of the individual clusters' indices.
- We put this average of averages into an array. We repeat for all rows and then finally find the mode of the array holding every row's average of each of its clusters' average index.
- Repeat for columns

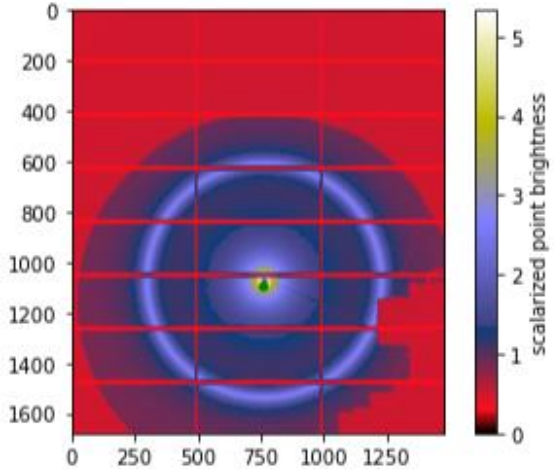
1

[462.0,440.89213304345964]



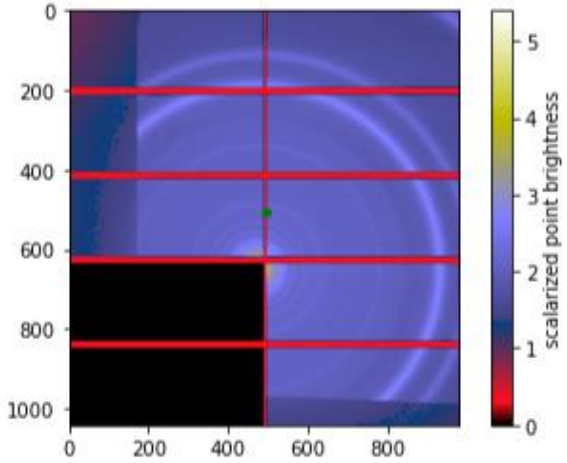
2

[758.5,1092.0748834774304]

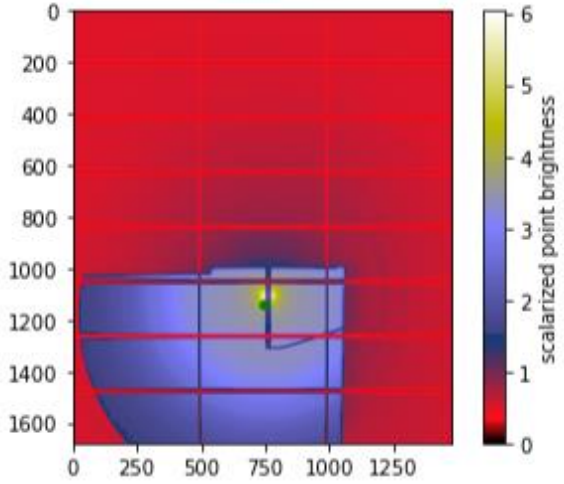


3

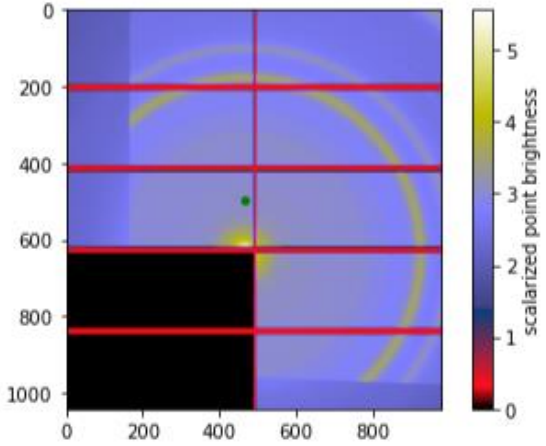
[494.5,504.24205237105383]



4 [742.5,1141.7670957981009]



5 [464.5,494.73253048340905]



There are two issues I am hoping to tackle by our Wednesday meeting. First, the columns are clearly off, while the rows are not (or at least not by much). I need to find out why. From quickly looking at the code, I can't see the reason.
Second, the run time for the column part of this algorithm is SIGNIFICANTLY longer than the row part. Sometimes around a minute longer (while rows is almost instant).
These two issues are likely caused by the same mistake, so solving one will likely solve the other, or at least put me on the track to solve the other.