

# General Notes

## Submitters

- Igal Spektor: 214146896
- Yosef Goren: 211515606

## The `sol.py` file.

The solution to the whole exercise is given in the `sol.py` file.

The parts of the exercise are separated and are in-order.

The `sol.py` contains all of our code and can run in many different modes - each corresponding to a specific part of a specific question.

For example - part C of question 1 is run with: `./sol.py q1_c` or `python3 sol.py q1_c`.

## Resource files

There are 2 types of resource files:

1. Provided resource files - the ones given by the exercise such as `Brad.jpg`  
`sol.py` assumes these files are found under the `..` dir with respect to the current working directory.
2. Our resource files - these are files we made on which our code depends. We provide these with our solution under the `our_resource_files` dir.  
`sol.py` assumes this dir will be found at `./our_resource_files`.

## Platform Compatibility

- You might have to install some pip packages to run `sol.py`, it should be quite a short list and all dependencies are found right at the start of `sol.py`.
- You will not be able to run `./sol.py` on windows and should instead use `python3 sol.py` or `python sol.py`.
- `sol.py` should work on both windows and linux, but only linux was checked.

## Question 1 - First Part

### Q1, Part a

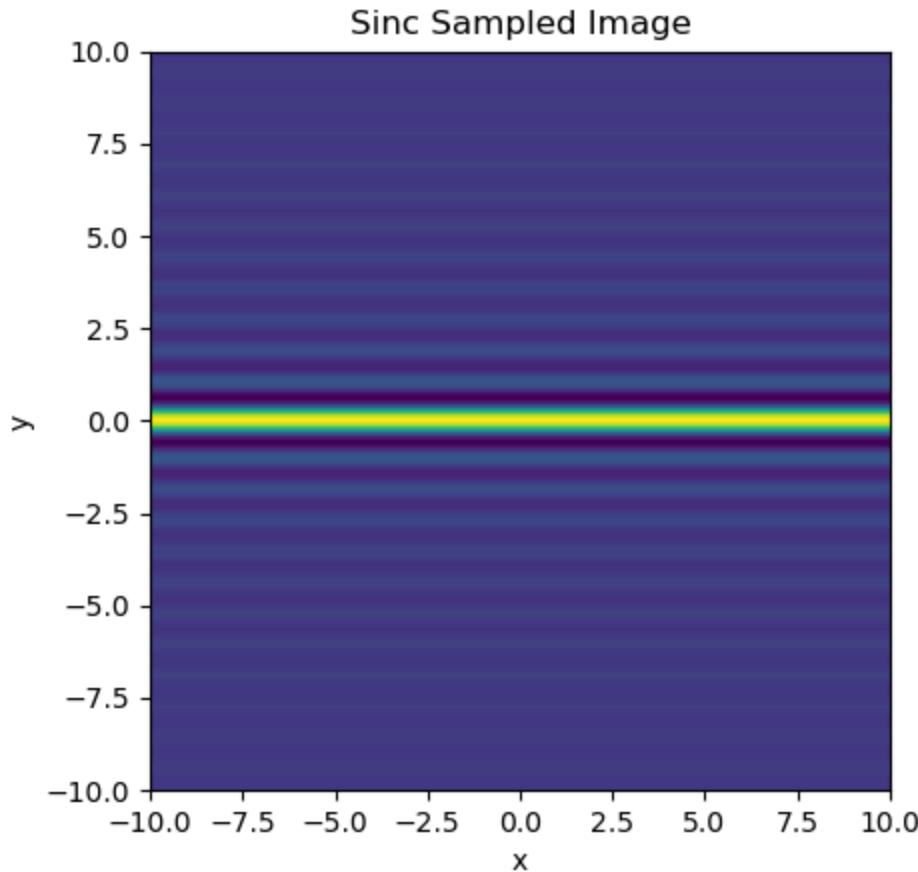
We implement sampling the image as specified within the `SincAnalysis` class.

### Q1, Part b

If we run:

```
./sol.py q1_b
```

We see:



We can see that the function is constant in the x axis, which matches the fact that our function's value does not depend on x.

We can see that the frequency in the y axis around the middle is something like 1.1, which is very close to what we would expect which is (where  $C_y$  is what is multiplied by  $y$  inside  $\sin$ ):

$$f_y = \frac{C_y}{2\pi} = \frac{\pi \frac{50}{21}}{2\pi} = \frac{25}{21} \approx 1.19$$

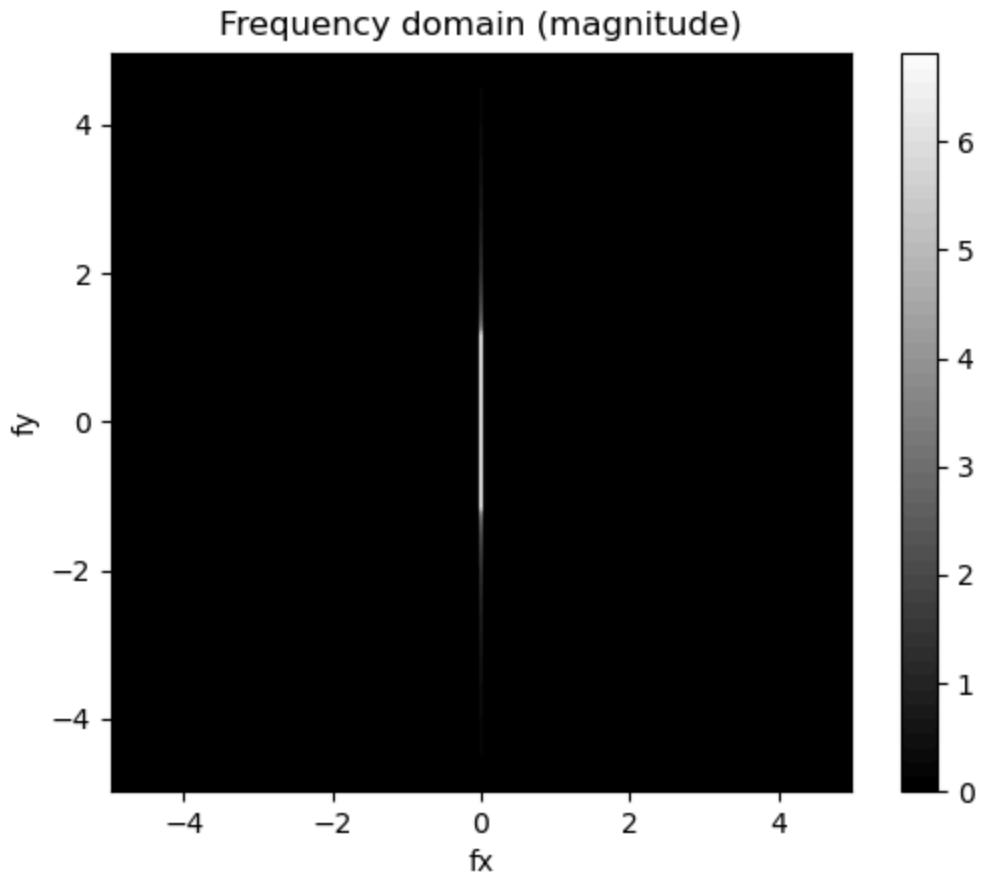
And this falls well within the margin of error my bad visual estimation.

## Q1, Part c

If we run:

```
./sol.py q1_c
```

We see:



This really matches what we expect: the bounds on the x frequency are almost 0 since it's just a constant, while the y frequencies are more spread out, while still being concentrated below 1.5.

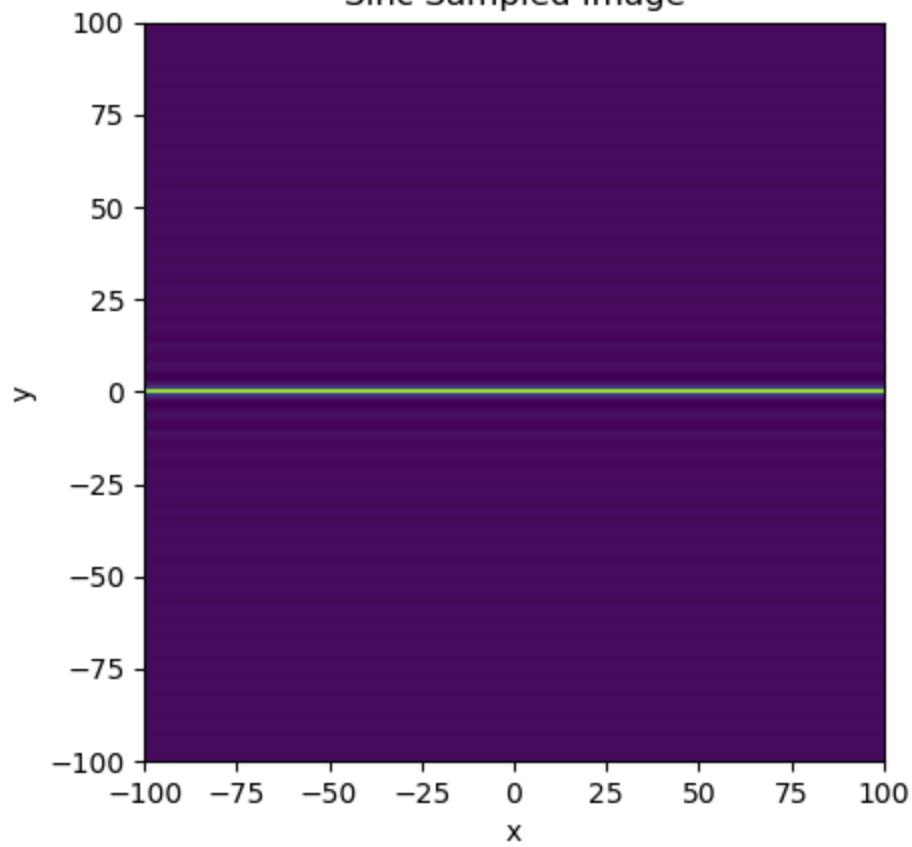
## Q1, Part d

If we run:

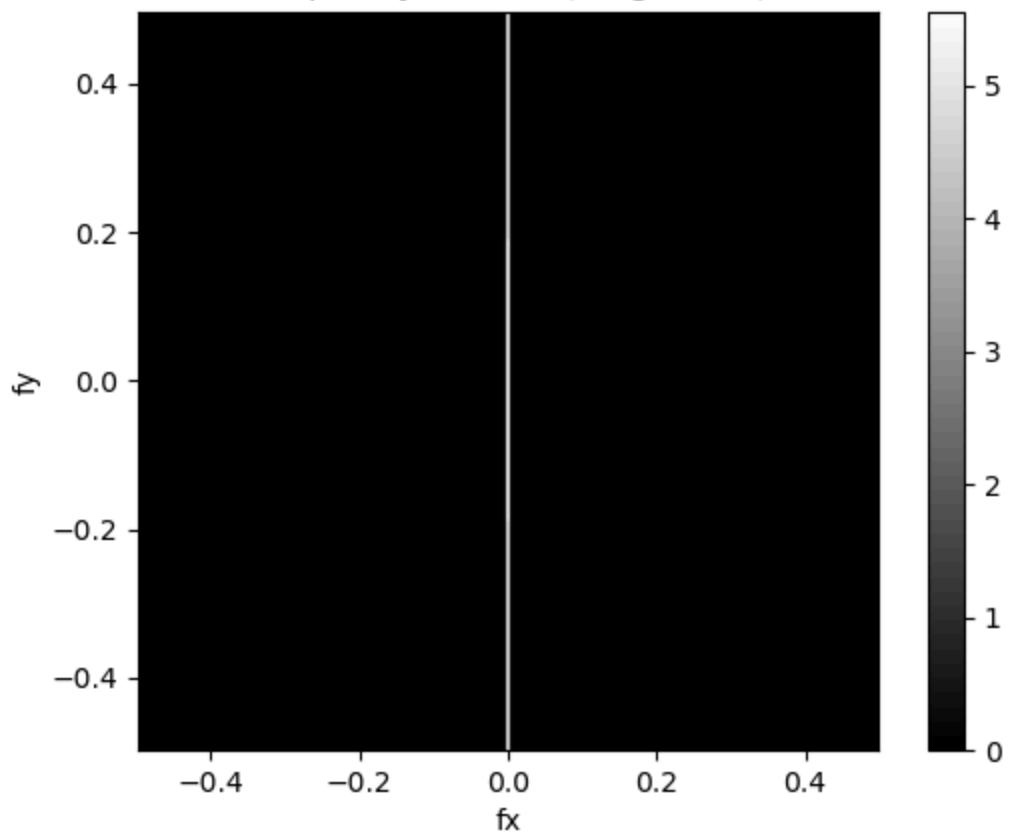
```
./sol.py q1_d
```

We see:

Sinc Sampled Image



Frequency domain (magnitude)



Interestingly, this time the frequencies are significantly lower than the real ones - it looks like a frequency of about 0.2.

This is ofcourse aliasing which happens since our sampling frequency is too low now.

## **Q1, Part e**

Yes. If we want to correctly sample the waves of the `sinc` function we will have to have a sufficient sampling frequency, which was clearly demonstrated in part d.

Theoretically, we would want our frequency to at-least match the nyquist bound:

$$f \geq f_{nyquist} = 2 \cdot f_{max} \approx 2 \cdot 1.19 = 2.38$$

Essentially, we want our y-axis sampling frequency to be at-least 2.38 while a sampling frequency of 0 or something really small should be fine for the x axis.

This matches the bounding box of the frequency domain images from earlier.

## **Question 1 - Second Part**

### **Q1, Part f**

Running:

```
./sol.py q1_f
```

Will display an animation.

Here is a screenshot from it:



It's spongbob running in circles around his burning house.

## Q1, Part g

This was more tricky than I expected. Here is what I did:

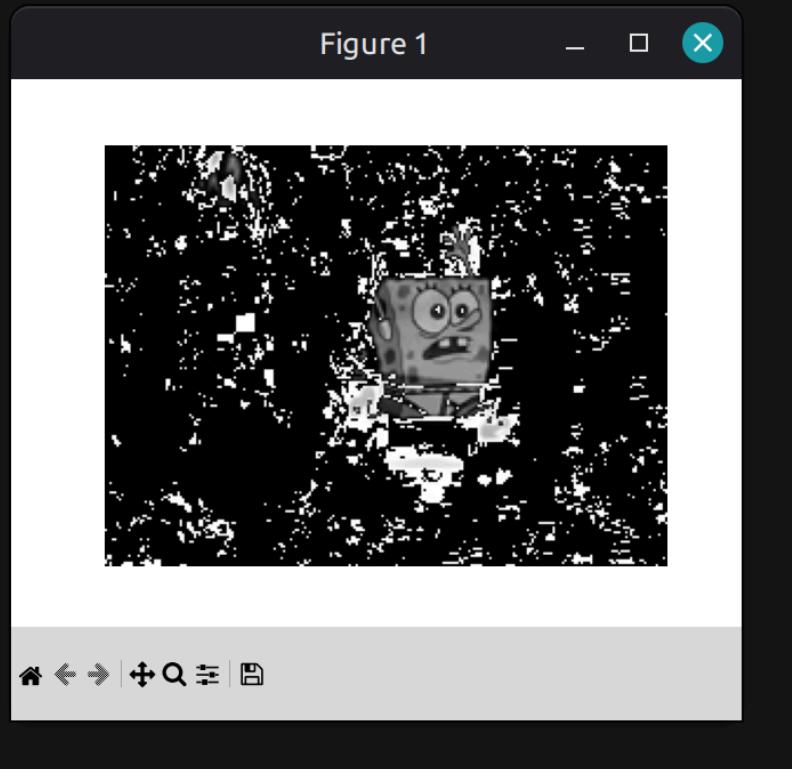
- First - diff the first frame from all frames and normalize the result: this makes it easy to tell which frame is identical to the first frame (when we see all black).
- Additionally, I added a feature for manually iterating through the frames with the arrowkeys so I can go through them one-by-one.

This can be run with:

```
./sol.py q1_g
```

So now I advance slowly through the frames...

```
o root@yogo-mint:~/projects/Digital_Image_Processing_2360860/Code1# ./sol.py showbob --diff-first --manual  
Manual mode: use keyboard to step through frames.  
Keys:  
- Right arrow / 'n' : next frame  
- Left arrow / 'p' : previous frame  
- 'q' : quit/close window  
  
Showing frame number: 1  
Showing frame number: 2  
Showing frame number: 3  
[]
```



And when I display frame at index 19 I see:

```
o root@yogo-mint:~/projects/Digital_Image_Processing_2360860/Code1# ./sol.py showbob --diff-first --manual  
Manual mode: use keyboard to step through frames.  
Keys:  
- Right arrow / 'n' : next frame  
- Left arrow / 'p' : previous frame  
- 'q' : quit/close window  
  
Showing frame number: 1  
Showing frame number: 2  
Showing frame number: 3  
Showing frame number: 3  
Showing frame number: 3  
Showing frame number: 4  
Showing frame number: 5  
Showing frame number: 6  
Showing frame number: 7  
Showing frame number: 8  
Showing frame number: 9  
Showing frame number: 10  
Showing frame number: 11  
Showing frame number: 12  
Showing frame number: 13  
Showing frame number: 14  
Showing frame number: 15  
Showing frame number: 16  
Showing frame number: 17  
Showing frame number: 18  
Showing frame number: 19  
[]
```



So indices strictly included in a single cycle are [0, 18] including edges - which is 19 different indices.

So there are 19[*Frames/Cycle*], if every frame was displayed for one second - spongebob would be cycling at 1/19[*Hz*].

## Q1, Part h

We would have to sample at least 2/19[*Samples/Frame*].

More explicitly, if we set:

- $F = \frac{1}{19} [\text{Cycle}/\text{Frame}]$
- $N = n[\text{Frame}/\text{Second}]$

We can find:

$$f_{max} = F * N = \frac{n}{19} \left[ \frac{\text{Cycle}}{\text{Frame}} \cdot \frac{\text{Frame}}{\text{Second}} \right] = \frac{n}{19} [\text{Hz}]$$
$$\Rightarrow f_{nyquist} = 2 * f_{max} = \frac{2n}{19} [\text{Hz}]$$

## Q1, Part i

Remember the different frames are [0, 18], so the image indices are equivalent (modulu %19). Since  $18 \% 19 = (-1) \% 19$ , sampling every 18 frames is the same as sampling the previous frame in terms of the sequence of frames that is shown.

I have added a variable `sampling_ratio` - which dictates 'every how many frames is the shown image updated?', so with `sampling_ratio=2` the image updates every 2 frames e.t.c.

If I run:

```
./sol.py q1_i
```

bob runs backwards.

If we were to run this with the default display interval of 20, he would run very slowly.

This is because while the sequence of images is identical to going back one frame in each cycle - each frame is now displayed 18 times longer.

So when we run `q1_i`, the display interval is instead set to 2.

In terms of frequency - this means cycling backwards is achieved by a sampling frequency of:

$$\frac{1}{18} \left[ \frac{Samples}{Frame} \right] = \frac{1}{18} \left[ \frac{Samples}{\frac{Cycle}{19}} \right] = \frac{19}{18} \left[ \frac{Samples}{Cycle} \right]$$

## Question 2

### Q2, Part a

This function was implemented as `interpolate_shift_dimension_fractional`.

### Q2, Part b

This function was implemented as `interpolate_shift_dimension`.

### Q2, Part c

If we run:

```
./sol.py q2_c
```

We see:



We have also added a mode where we can play with the shift interactively by click and dragging with the mouse.

This can be done by running:

```
./sol.py q2_interactive
```

Which will show something like:

Click and drag to shift image. Press q to quit.

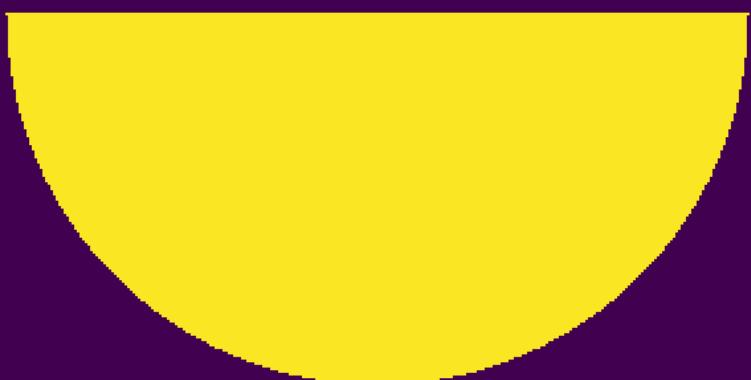


## Q2, Part d

If we run:

```
./sol.py q2_d
```

We see:



## Q2, Part e

If we run:

```
./sol.py q2_e
```

We see:

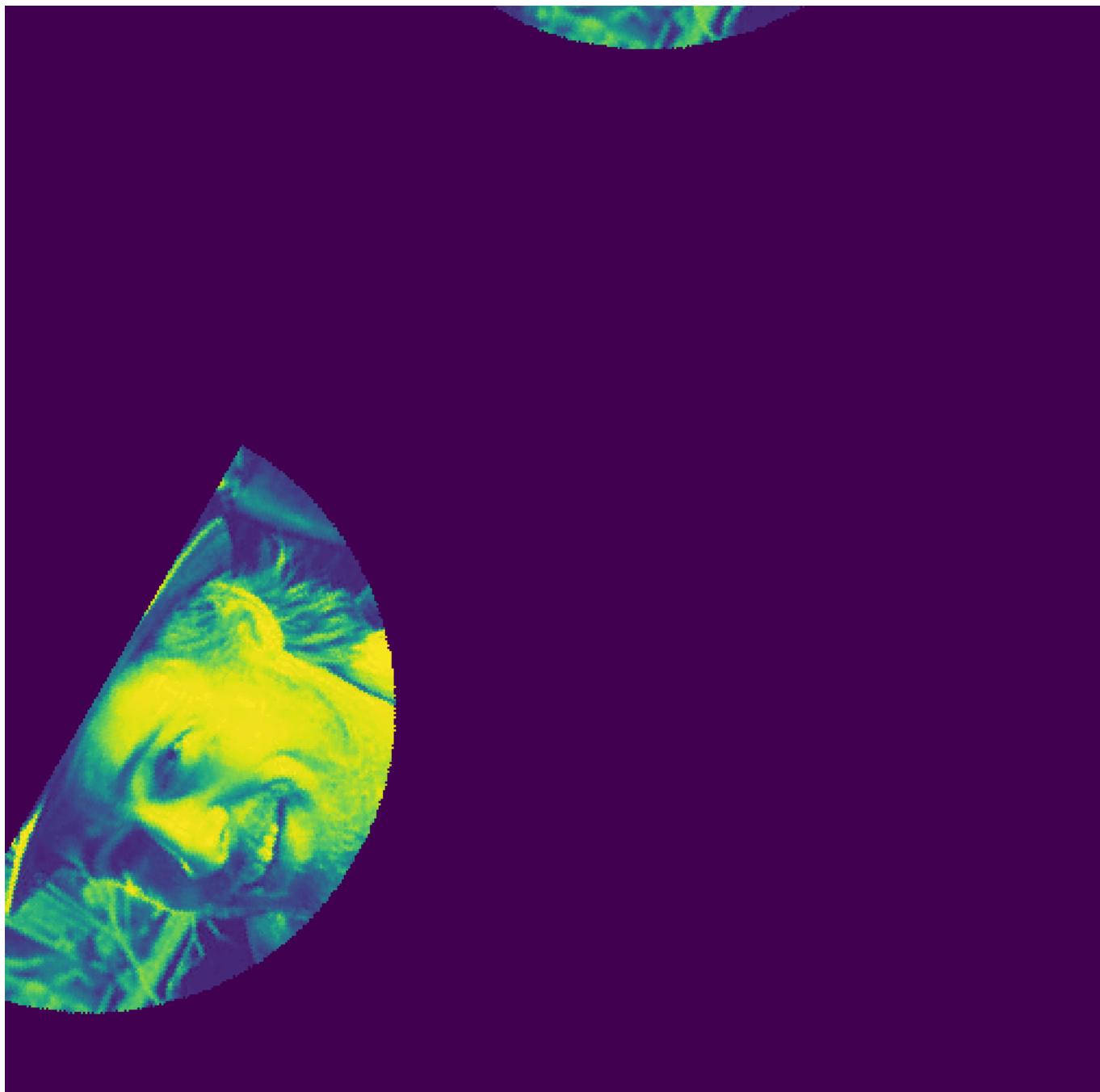


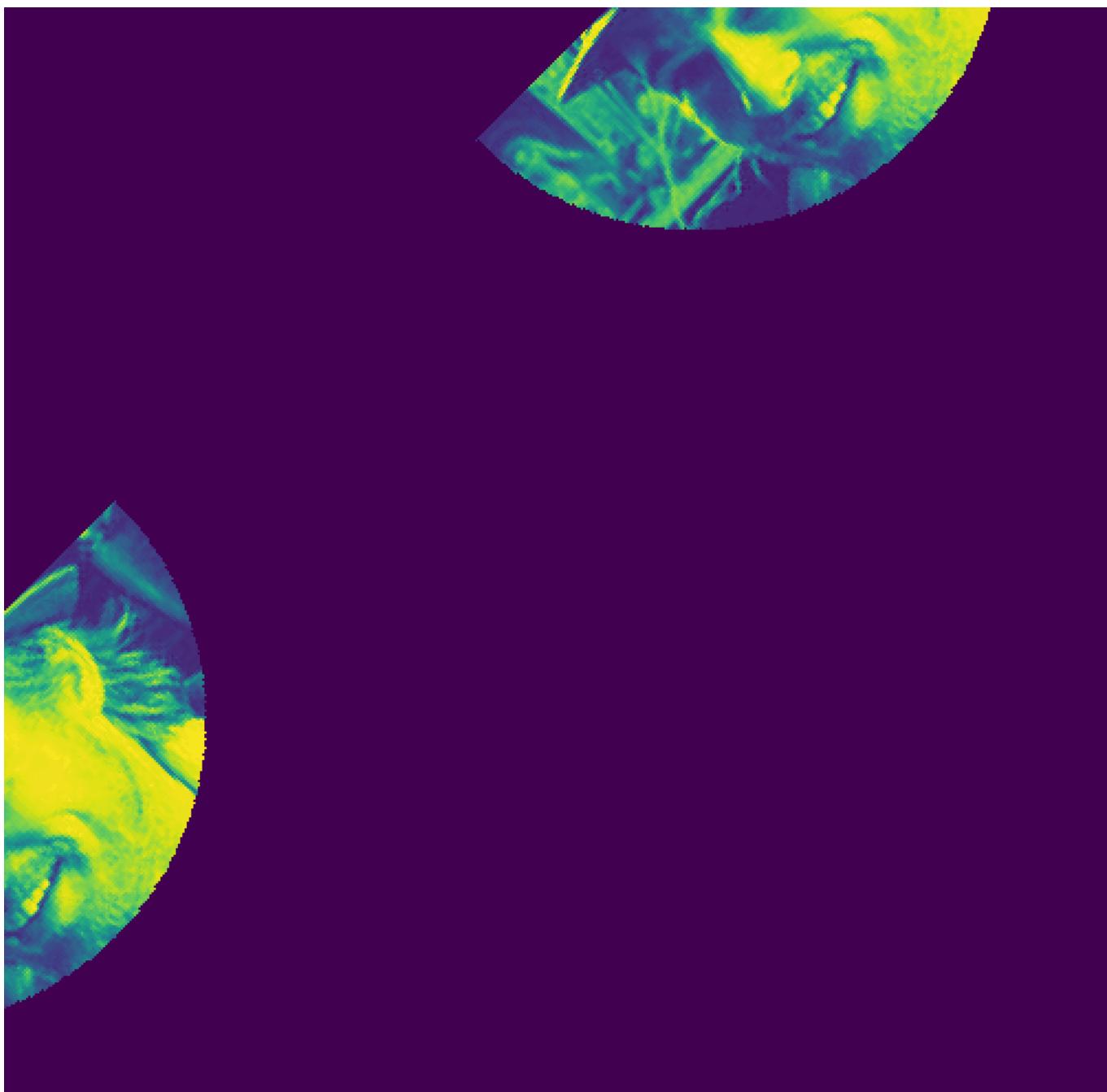
## Q2, Part f

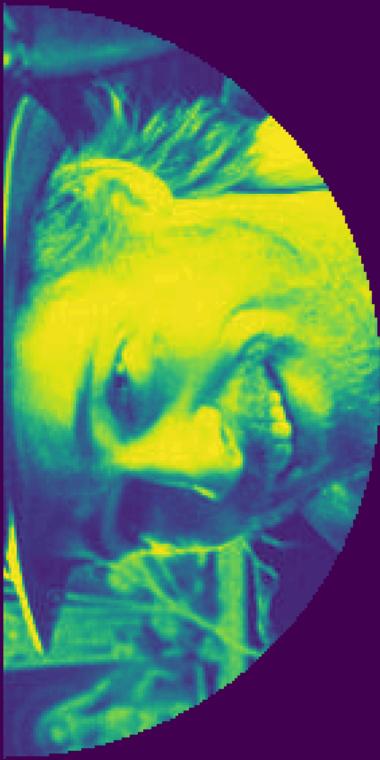
If we run:

```
./sol.py q2_f
```

We see:







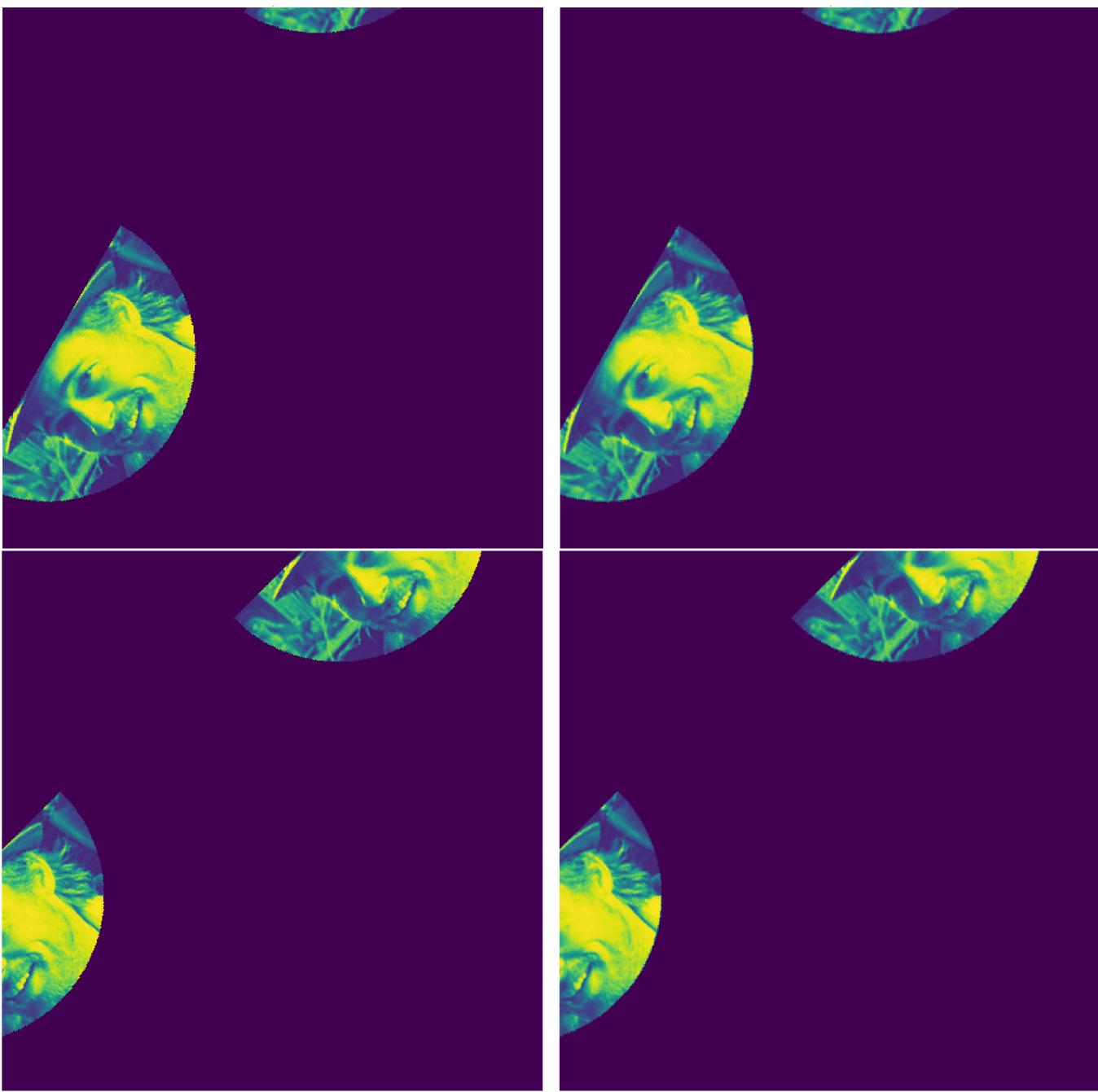
It looks like the edges in the rotated images are very 'sharp' - the image lost it's smoothness in this process.

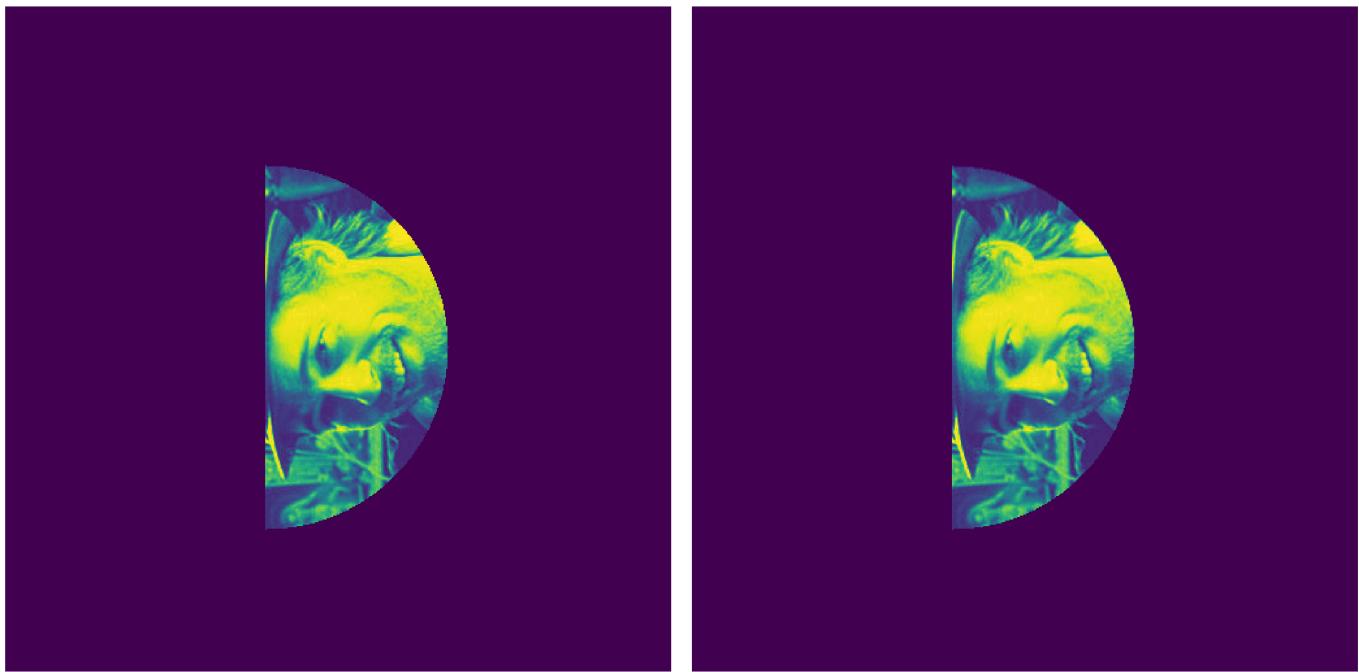
## Q2, Part g

If we run:

```
./sol.py q2_g
```

We will see the interpolated and non-interpolated images side-by-side for each angle:





This yielded a significant improvement in the issue we described before - but not for the 90 degrees case - since it did not have much of an issue to begin with.

We think that the interpolated versions are better in the cases where it matters.

It should not matter in the case of all right angles - 0, 90, 180, 270, since there is no interpolation to be done in these cases - the pixels fall 'right in place' of others.

## Question 3

### Q3, Part a

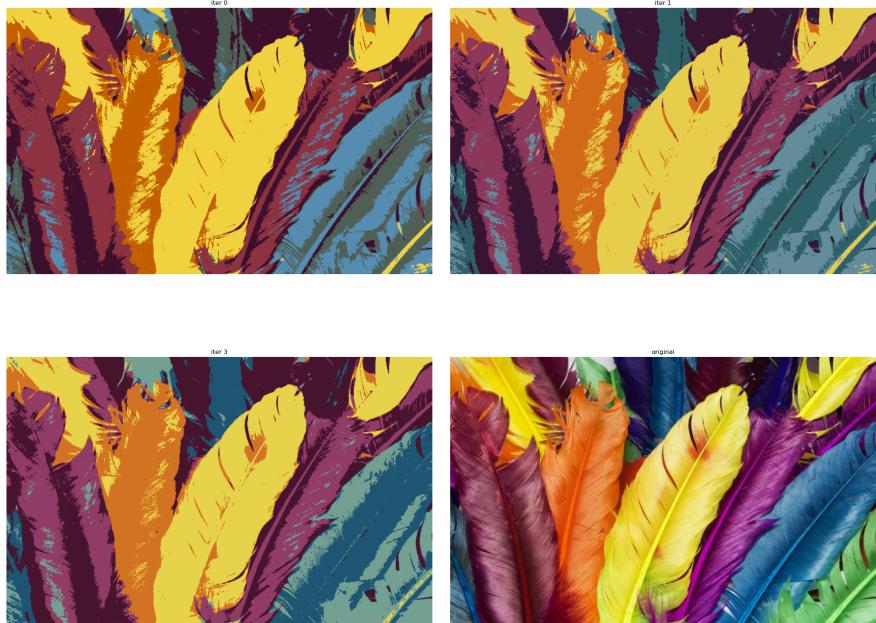
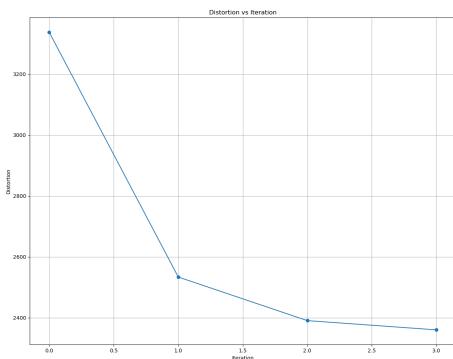
The function was implemented as `max_lloyd`.

### Q3, Part b

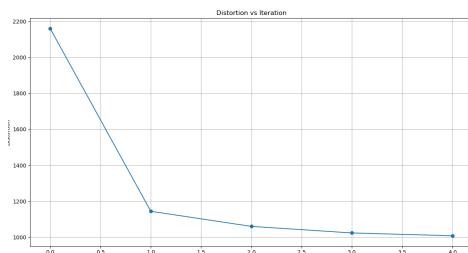
If we run:

```
./sol.py q3_b
```

For 6 colors we see:



And for 15 colors we see:



Both visually and numerically - we can see that 15 colors yielded a much nicer result - even at the same amounts of iterations.

## Q3, Part c

Since the described algorithm is not deterministic - each time we run it - we are likely to get a somewhat different result.

This comes in to play both when we randomize the initial vectors, and in how we handle vectors to which no pixel was mapped (we randomize a different vector to replace it).

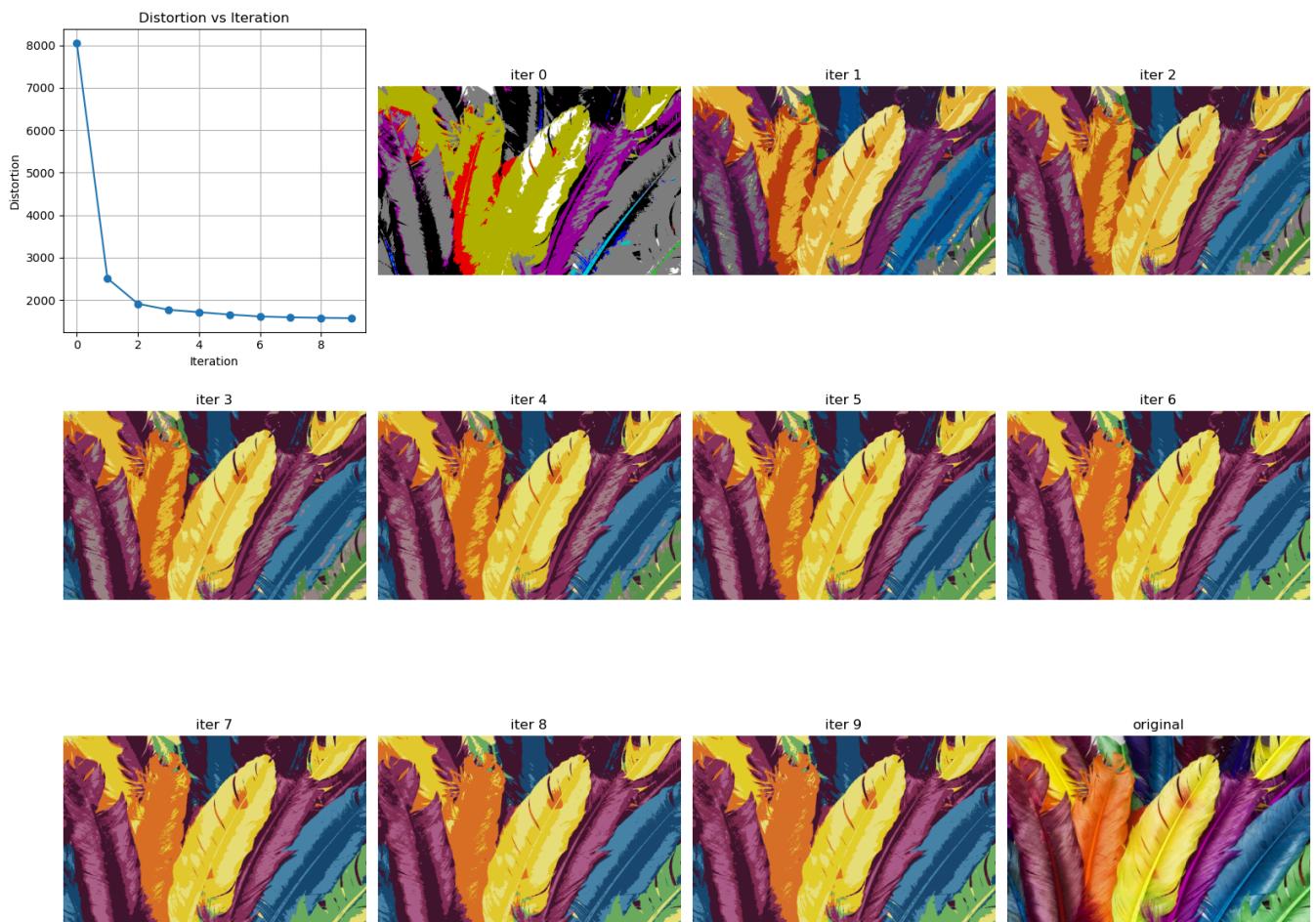
Yet, in our specific implementation the randomization is controlled by a seed that may be specified, so this pseudo-randomness can be controlled to achieve deterministic results.

## Q3, Part d

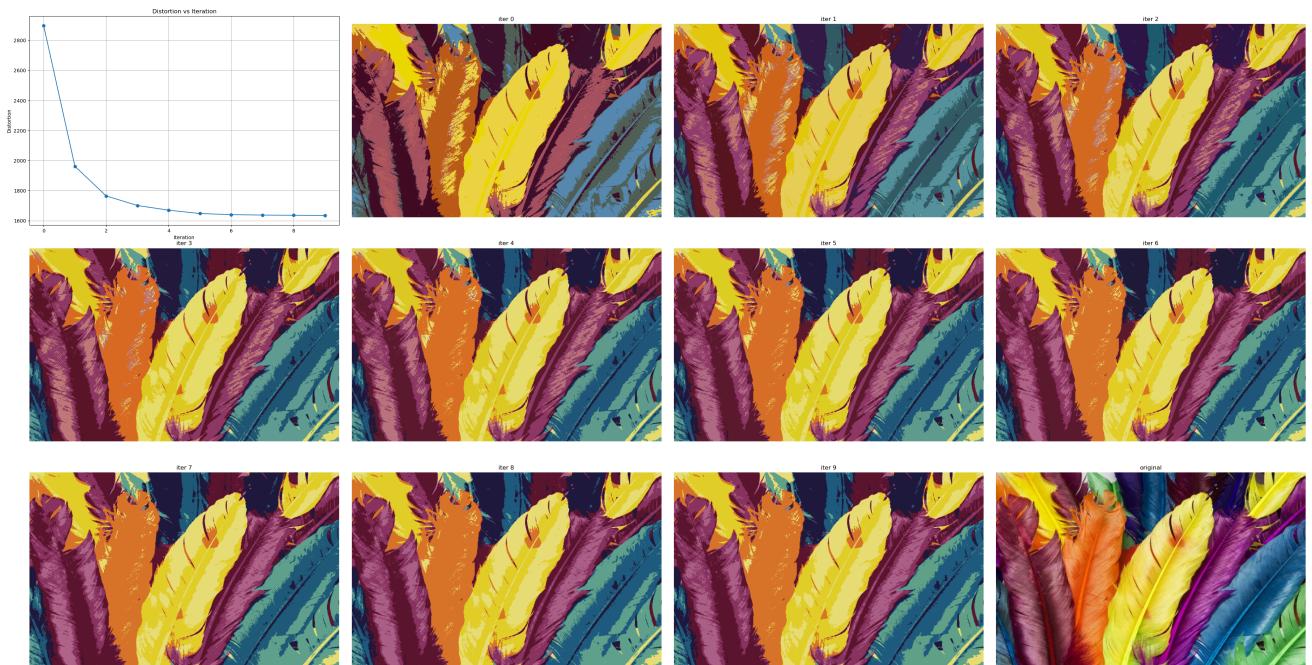
If we run:

```
./3.py q3_d
```

We first get the results for the pre-defined initialization vectors found in `max_lloyed_iv.json` (specified by the question):



And then get the result for if we randomize the same number (9) of initial vectors instead:



So it looks like:

1. the initial distortion with our specified set of vectors is much higher.
2. after many iterations, they converge to very similar results (both visually and numerically).

So it looks like just guessing randomly did much better than hardcoding.

It's not impossible to provide initial vectors that would be better, but it seems like it's not a trivial task either as guessing initial vectors randomly performs better than expected.

## שאלה 4

א. פונקציית הספריה shannon\_entropy מחשבת את האנטרופיה עבור תמונה. קיבלנו אנטרופיה של 7.011. תחילה נחשב את האנטרופיה המקסימלית האפשרית עבור תמונה עם 8 ביטים כלומר 256 ערכים:

$$max = - \sum_{i=1}^{256} \frac{1}{256} \cdot \log_2 \frac{1}{256} = 8$$

לכן קיבלנו אנטרופיה 7.011 שזה קרוב למקסימום שהוא 8. لكن ההתפלגות של רמות האפור בתמונה היא כמעט שווה, ככלומר אנחנו משתמשים בכל רמות האפור באופן שווה כמעט לכל רמות האפור (אבל לא בדוק, אחרת האנטרופיה הייתה 8).

.ג.

לפי החישוב בסקריפט, אורך הקוד הוא 460977, יהו הדחיסה

$$\frac{8}{huffman\_avglen} = \frac{\text{uncompressed bitrate}}{\text{compressed bitrate}}$$

זהו יהו דחיסה נמוך אבל זו תוצאה הגיונית שכן לפי תוצאה מההרצאה:

$$entropy \leq huffman\_avglen \leq entropy + 1$$

זהו אכן מתקיים כי

$$7.011 \leq \frac{8}{1.1374} < 8.011$$

.ד.

פענחנו וקיבלנו  $MSE = 0$ . תוצאה מאוד הגיונית וצפוייה כי קידוד האפסן הוא קידוד lossless שכן ניתן לשחרר את התמונה באופן מושלם ולקבל שגיאה שווה ל-0.

.ה.

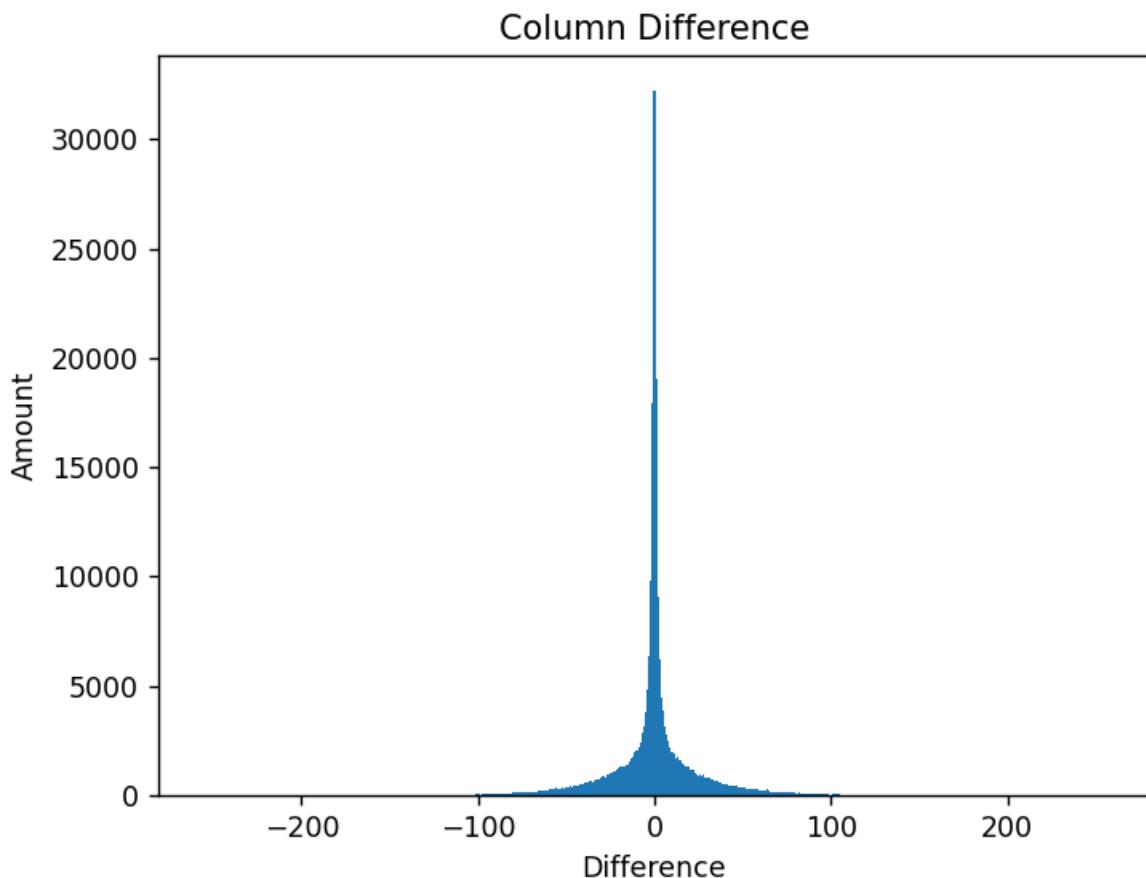
קיבלנו אנטרופיה ששויה ל-2.1453, תוצאה הגיונית בהתחשב בכך שהדגל של מאוריציו הוא 4 מלבנים אופקיים כך שכל מלבן בעל צבע אחד, אך אנחנו משתמשים בכמות מצומצמת מאוד של רמות אפור מה שמוריד את האנטרופיה מאוד.

קיבלנו קצב מידע השווה ל $2.3125$ . לפי התוצאה מההרצאה מסעיף ג', הגבול התאורטי עבור קצב מידע של האפן הוא  $3.1453$  (שזה האנתרופיה ועוד אחת). קצב המידע שקיבלנו קרוב מאד לאנתרופיה لكن הקידוד הוא מאד עיל.

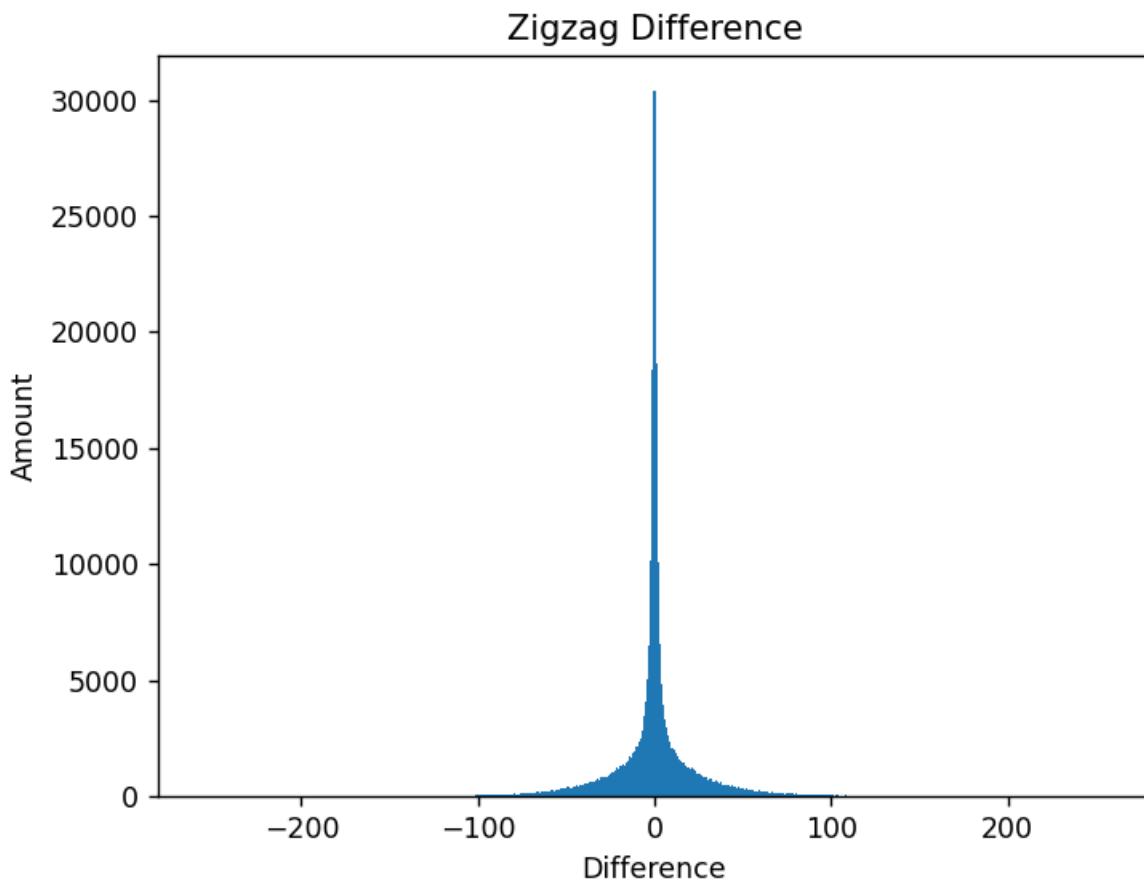
ו.

להלן ההיסטוגרמות:

ההיסטוגרמה עבור סידור עמודה:



ההיסטוגרמה עבור סידור זיגzag:



התוצאות של היסטוגרמות נראות זהות כמעט לחלוטין. זה מצופה, כיוון שהתמונה היא טבעית لكن יש לה יתרות מרחבית רבה (נוף של סקוטלנד הוא בוודאי טבעי), על כן במעט אנחנו מצפים פיקסלים קרובים להיות בעלי אותו ערך פחות או יותר. لكن גם עבור סידור עמודה וגם עבור סידור זיגזג אנחנו מקבל התפלגות לפולסיאן "יפה" כמו שאנו רואים בהיסטוגרמות. לא משנה אם הסידור או עמודה או זיגזג, בכל סידור, ברוב המקרים פיקסלים קרובים בסידור הם גם קרובים בתמונה (בסידור עמודה אנחנו לוקחים ברוב המקרים פיקסלים קרובים מאותה עמודה, ואילו בסידור זיגזג לוקחים ברוב המקרים פיקסלים קרובים מאותו אלכסון).

קצב המידע עבור סידור עמודה: 5.992

קצב המידע עבור סידור זיגזג: 5.970

קיבלנו כי סידור זיגזג הוא מעט יותר טוב, כי יש לו קצב מידע נמוך יותר. הסיבה לכך היא זיגזג הוא יותר טוב זה כי הוא מנצל את היתירות המרחבית יותר טוב. סידור עמודה עבור על העמודות של התמונה ובכל סיום של עמודה הוא "קופץ" לעמודה הבאה, מה שהורס

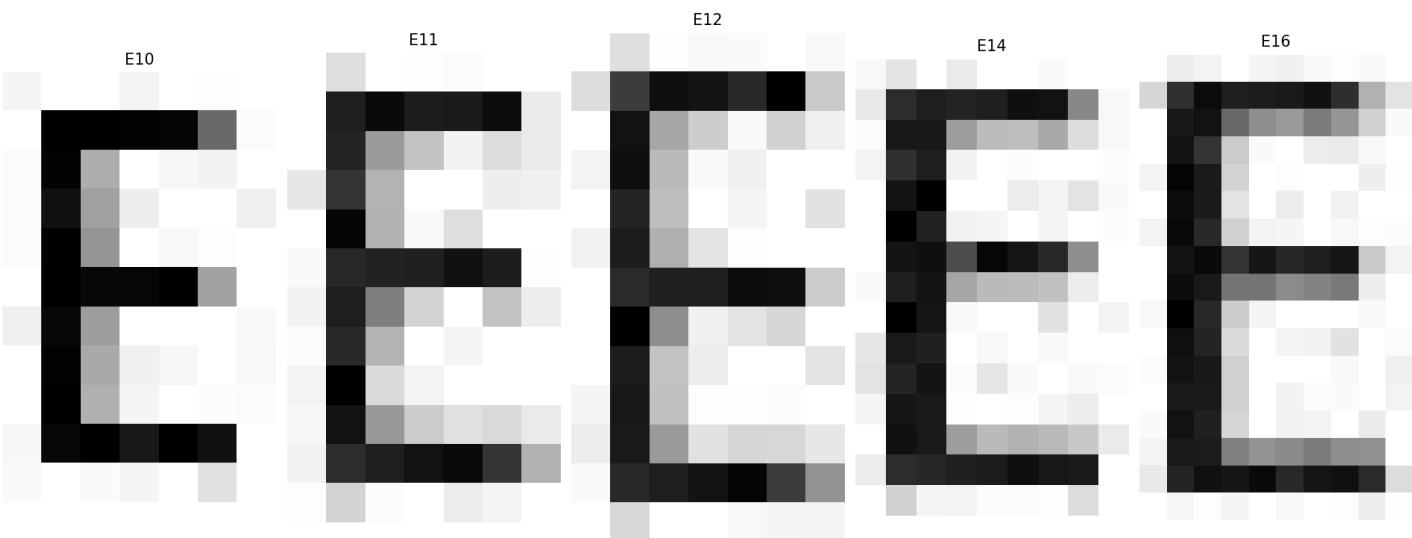
את הטעע הדו מימדי של התמונה. לעומת זאת, סידור זיגzag עבור אלכסונים ולא קפיצות لكن הסיכוי לקבל שני פיקסלים עם רמת אפור דומה הוא יותר גבוה.

ז. כן. למשל, תמונה שהיוא רוש לא משתמשת לקידוד באמצעות תכונות הפרשים מכיוון שהיא לא טבעית ואין לה יתרות מרחבית, ולפיכך ההיסטוגרמה של תמונה הפרשים לא תהיה לפולסיאנית. וכך אנו כפלנו את התמונה (במקום 256 רמות אפור נקבל 511 רמות אפור) וכעת התמונה החדשה צריכה יותר מקום בזיכרון ללא תועלת.

## שאלה 5

.ב.

להלן חמישה תמונות ייחודי:

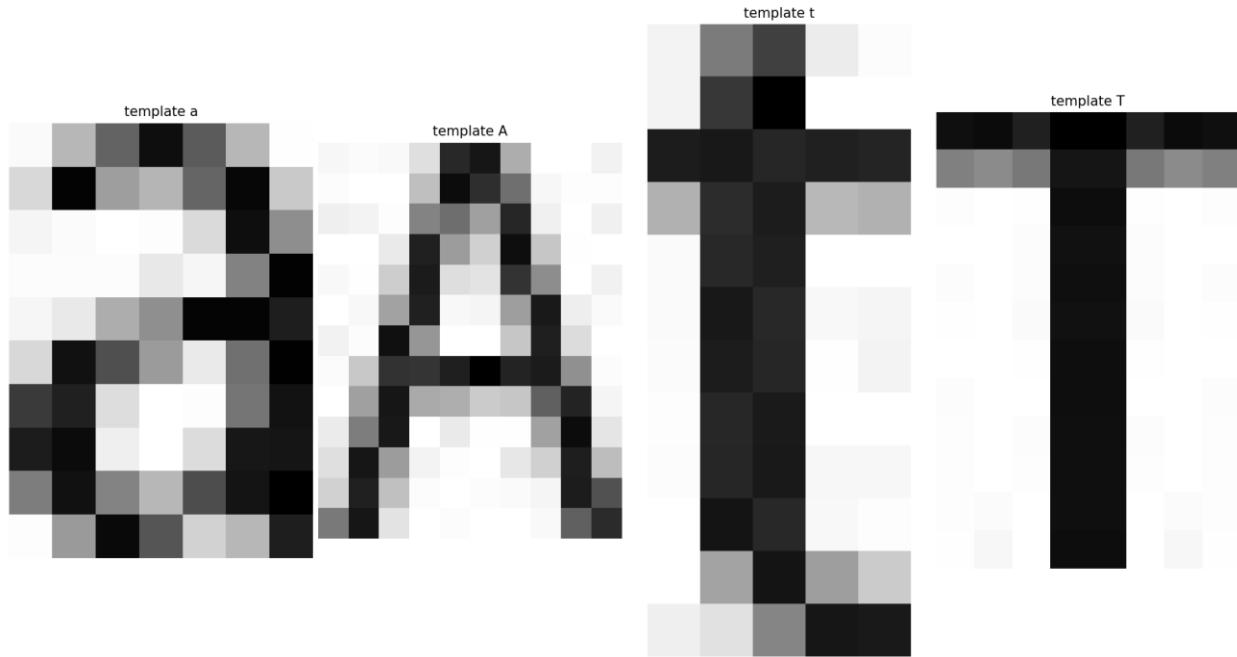


לפי האלגוריתם שלנו, כובלנו את ה-HSS הכי נמור עבור  $E14$ , שכן גודל הגוף הוא 14 לפי האלגוריתם.

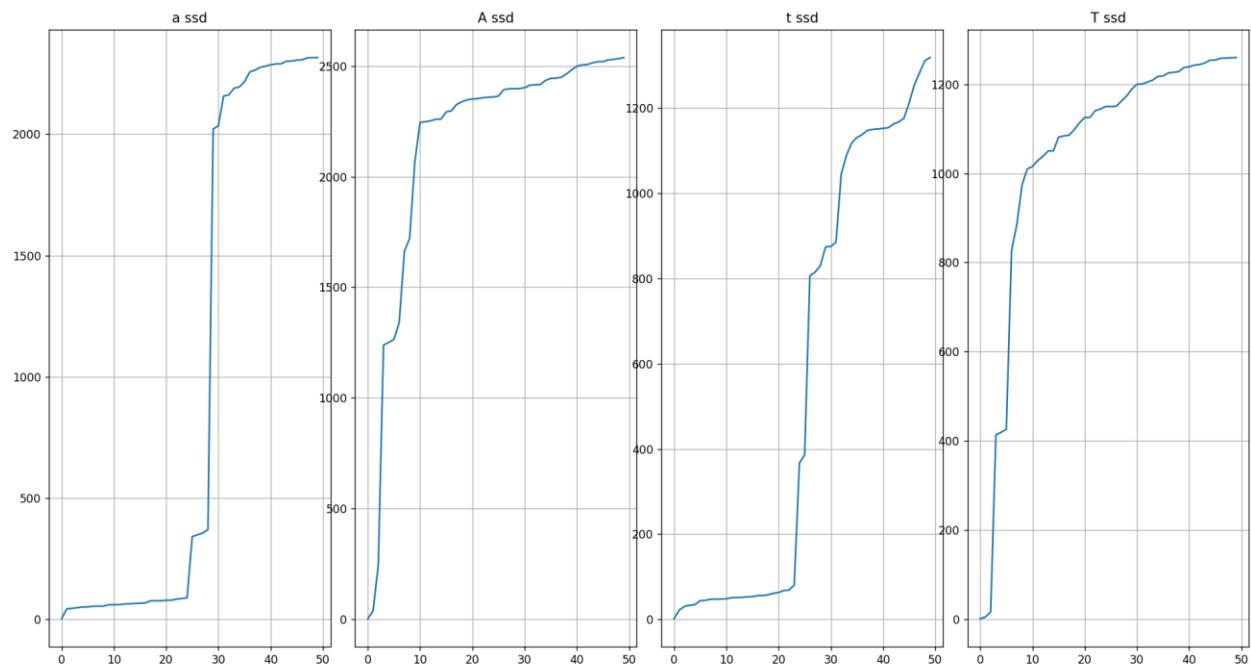
मבחן ידנית שלי, השוותי את גודלי הגוף לטקסט ומצאת שגודלו הגוף הוא אכן 14 וכן האלגוריתם נכון.

.ג.

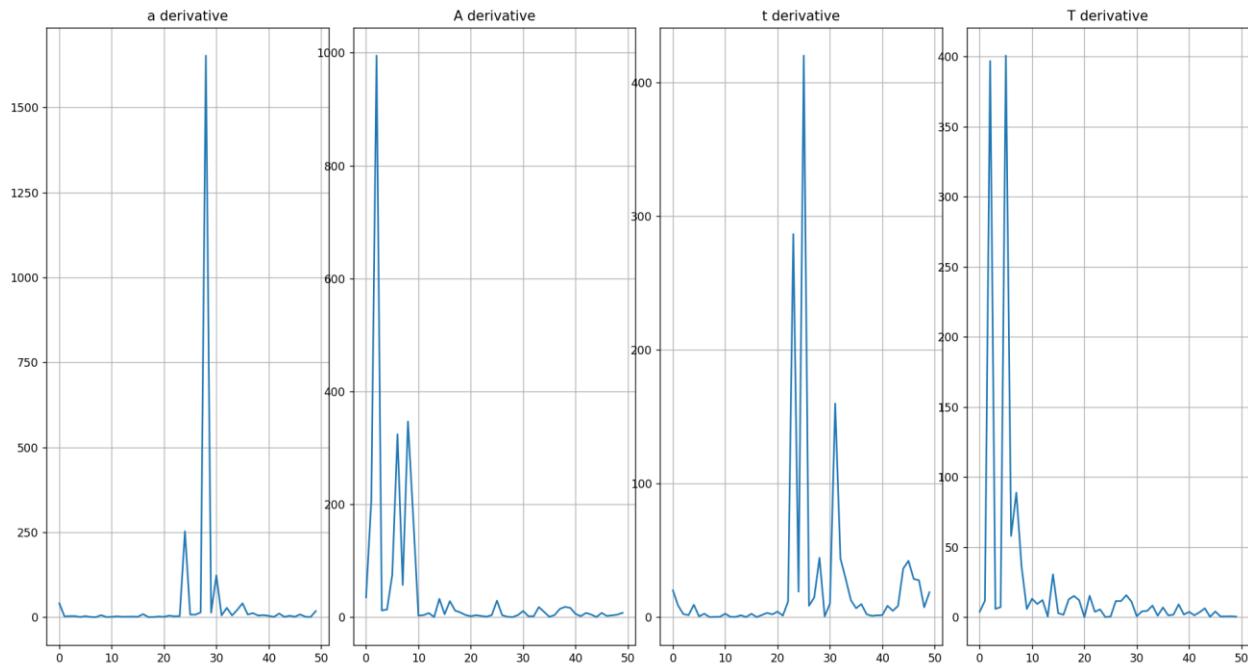
להלן התבניות שבחרנו:



בחרנו את הסוף באופן הבא: לכל תבנית חישבנו את  $SSD$  (עם נרמול על סמרק גDAL).  
תמונה אחת, כי לכל תבנית יש גודל שונה), לאחר מכן מינינו את וקטורי  $SSD$  מהקטן לגודל.  
עבור התאמות אמרו להיות  $SSD$  נמוך ובעור חוסר התאמות אמרו להיות  $SSD$  מאד  
גבוה. لكن בדקנו את הגרפים של  $SSD$  המומין:



ניתן לראות שיש קפיצות מאוד גבוהות בשלב מסוים, לכן בדקנו את גרפ הנגזרות של הSSD הממוני:



בחרנו סף שווה ל270, מכיוון שבפעם הראשון יש קפיצה מאוד גבוהה בנגזרת, איז הנגזרת נהיית יותר גבוהה מ0.270. כאשר יש קפיצה צו בערך הנגזרת, זה אומר שאין יותר התאמות ולכן שגיאת SSD נהיית גבוהה מאוד.

כלומר, התנאי שלנו הוא: האינדקס הראשון במערך הנגזרות של וקטור SSD הממוני, שמקיים כי גודל ערך הנגזרת גדול מ0.270.

קיבלנו את התוצאה הבאה:

Occurrences of a: 29

Occurrences of A: 3

Occurrences of t: 24

Occurrences of T: 3

בטקסט באמת יש 3 פעמים t, 3 פעמים T, 24 פעמים a. אבל יש 27 מופעים של a ולכן האלגוריתם ספור את a פעמיים יותר מאשר צריכה. יתכן שהזיה קריה כי אנחנו רואים בגרף הנגזרות עבור a שהייתי קפיצה קטנה בערך באינדקס 13 שנחננו התעלמנו ממנו, ולאחר מכן מכך קפיצה מאוד גבוהה באינדקס 29, אך האלגוריתם מתקשה להתאים את

התבנית לאות a בטקסט בכל מהמקרים. לדעתי הסיבה לכך היא שבניגוד לגוף הטקסט, הכותרת והחתימה הם מעט יותר מודגשים מה שמקשה על האלגוריתם לזהות את כל האים באופן מדויק.

עבור עברו mr smoke too much להלן הטקסט החדש:

Replaced c with k

### Epigram For Wall Street

I'll tell you a plan for gaining wealth,  
Better than banking, trade or leases —  
Take a bank note and fold it up,  
And then you will find your money in kreasess!  
This wonderful plan, without danger or loss,  
Keeps your kash in your hands, where nothing kan trouble it;  
And every time that you fold it akross,  
'Tis as plain as the light of the day that you double it!

Edgar Allan Poe

התמונה נראה מאד טبيعית, נראה כאילו כל a שהחליף את c נמצא במקומו באופן טבעי  
ואין בעיות של alignment.