TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TOMI PALOVUORI

AUTOMATED CLUSTERING OF PERTURBATION-INDUCED QUANTUM SCARS

Bachelor’s Thesis

Examiner: Joonas Keski-Rahkonen

ABSTRACT

Tomi Palovuori: Automated clustering of perturbation-induced quantum scars
Tampere University of Technology
Bachelor’s Thesis, 23 pages, 3 appendix pages
May 2018
Bachelor’s Degree Programme in Information Technology
Major: Machine Learning
Examiner: Joonas Keski-Rahkonen

Keywords: quantum scars, image clustering, hierarchical clustering

Some of high-energy eigenstates of a highly disordered quantum system are strongly scarred by periodic orbits of the unperturbed classical counterpart. Detecting these scars has thus far relied on a researcher to go through individually thousands of solved eigenstates of a perturbed system by eye. However, the probability densities of the eigenstates can be seen as two-dimensional images, and the classification become an image clustering task. With a bit of preprocessing the image clustering problem is generalized for any set of scars. Utilizing a simple comparison function for calculating the similarity between any two scars a distance matrix can be calculated for hierarchical clustering.

The comparison function for scars with differing scale and rotation simply contains a comparison for each size and rotation possible in the dataset. Within the dataset received all scars are centered and bound by circles surrounded by low noise, so they can be scaled by resizing the picture and either removing or padding the outlines. Squared error is the best method of calculating the distance between scars. A weak gaussian blur greatly improves the results.

Hierarchical clustering with SciPy includes multiple clustering methods and distance metrices. Best metric for clustering the scars is calculating correlative distances from the comparison function’s distance matrix. Of clustering methods both UPGMA and WPGMA cluster correctly, but the results of UPGMA are easier to extract. Mean images can be calculated from aligned clustered images to graphically present clustering results of any number of clusters. Accurate mean images require more comparisons to be calculated but stored information from the distance matrix can be used to reduce the number of required comparisons.

Wanted classes of scars are clustered best at different points of clustering. The last three clusters for the test set include clear bouncing balls, circles and the rest. Within the rest there are noisy bouncing ball -like scars with a threshold distance of 21 and pentagram-scars with a threshold distance of 16-18. Threshold of 17 results in 9-12 classes, of which 3-4 are bouncing balls, 2-3 are circles and one is pentagrams. There are classes of quantum scars not introduced in the test set and the thresholds for those must be examined separately.

Classification from clustering must still be done by hand or alternatively a simple query-function can be used. Using simple class templates developed compare()-function can be utilized to classify scars. However, this does require prior knowledge about possible classes inside a given set. Classification of about a dozen images by hand is still significantly faster than the classification of the entire 2000 image set.

PREFACE

This thesis is the result of collaboration by the Laboratory of Signal Processing and the Laboratory of Physics in Tampere University of Technology. I would like to thank the examiner of this thesis Joonas Keski-Rahkonen for helping with the physics part, of which I still understand very little to nothing about. I would also like to thank professor Joni Kämäräinen for being my contact in all matters regarding signal processing and suggesting the general concept of calculating distances and hierarchically grouping them. The faculty and people in charge of my programme also deserve to be thanked. It took some special treatment towards me to allow the completion of this thesis after just little over two years of studies. I am also grateful to a good friend of mine Jimi Järvelin, who read through this thesis marking numerous grammatical errors. A lot of the work involved with this thesis also took place away from home, so a big thank you for my girlfriend Heini Turunen for not holding it against me.

Tampere, 06.05.2018

Tomi Palovuori

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

Cv2	An image processing library for Python
NumPy	A Python library including matrixes and operations for them
PIL	Python Imaging Library
SciPy	A Python library including functions for rotating and clustering
TUT	Tampere University of Technology
*	Multiplication in Python
/	Division in Python
[x,y]	A closed value range from value x to value y

1. INTRODUCTION

At microscopic level, nature is governed by the principles of quantum mechanics. A key concept in quantum mechanics is a complex valued quantity, known as a wave function, which is determined by the Schrödinger equation. The physical interpretation for a wave function is that its squared norm represents probability density for a quantum particle. [1]

It was assumed that each of the eigenstates of a classically chaotic system would be featureless and random. However, a quantum scar is a striking visual example of quantum suppression of classical chaos: probability density of some eigenstates is enhanced around short unstable periodic orbits of the corresponding classical system. The probability increase is so significant that the occurrences of scarred eigenstates may equal to probabilities over 50% if strong enough scarring is present. [2] [3]

A new kind of quantum scarring, called perturbation-induced quantum scarring, was recently discovered by researchers at Tampere University of Technology (TUT), Tulane University, Harvard University, and MIT [4]. In this artificially produced scarring, a separable, two-dimensional system is perturbed by potential bumps resulting in eigenstates scarred by classical orbits of the corresponding unperturbed system. Combined with the high controllability and enhanced wave packet propagation, these scars may be employed in quantum transport [5].

The perturbation-induced scarring is studied numerically by solving thousands of the eigenstates of a perturbed system in a simulation. This leads to massive amounts of data on two-dimensional probability densities related to the solved wave functions. Only some of the eigenstates are scarred, and thus far detecting them has relied on a researcher going through the states by hand. Software methods to cluster or classify quantum scars have been tried with self-organizing maps and convolution neural networks, but the results have not been satisfactory in grouping scars with ranging scale and rotation.

The purpose of this thesis is to provide a simple and streamlined albeit computationally more consuming approach for the Laboratory of Physics at TUT. By developing a comparison function that returns a scalar value for the similarity of two scars, different clustering methods can be used to group similar scars without the need of a learning dataset. Comparison function also allows the development of a query-based classification using a small set of different quantum scar class examples.

This document is structured as follows. Chapter 2 explains briefly the format in which quantum scars are presented in the dataset, their normalization and classification by eye.

Chapter 3 presents possible ways for calculating the similarity between any two scars regardless of rotation and scale. Chapter 4 discusses the selected methods for the comparison function and explains why those were selected as well as some results from the dataset and the calculative load of using the function. Chapter 5 includes calculation and clustering of distance matrices and the methods and metrics used to achieve the best outcomes as well as means to calculate mean images of clusters. Chapter 6 presents results of clustering and mean images for three subsets of the test set. Chapter 7 draws conclusions of results and discusses the future applications of the software. Software built for this thesis is written in Python. The figures of quantum scars in this thesis employ a logarithmic color scale where darker intensities equal higher probability densities. These figures are further processed for better printing on paper.

2. QUANTUM SCARS AS IMAGES

Perturbation-induced quantum scars are enhancements of probability density in the eigenstates of a perturbed quantum system that occur around periodic orbits of the corresponding unperturbed classical system. In this thesis, the probability density related to an eigenstate is viewed as a two-dimensional image. Resulting images of thousands of numerically solved eigenstates may include scars which resemble classical orbits in the unperturbed system and random fluctuations referred as "noise" later. Figure 1 shows an example image of a scar for which the classification software is aimed. The scar is bound inside a circle and surrounded by white noise with a magnitude at least a million times lower than values inside the circle.

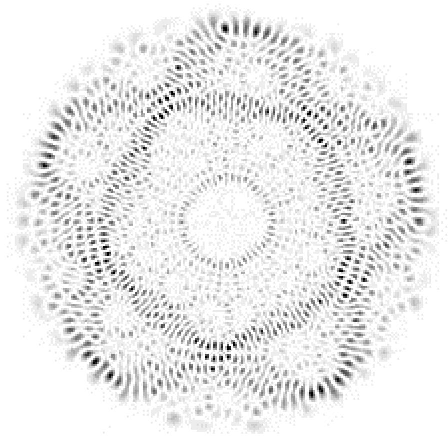

Figure 1. *A quantum scar represented as an image.*

There are only a few larger classes of scars within the received testing set. The ideal grouping for scars would be branding all images with only noise to one class and separating the rest carefully. Excluding fully noise-like images, there are three main classes within the data set: a bouncing ball, a pentagram and a circle. Bouncing balls are quantum scars which appear as strong lines describing bouncing motion of a classical particle. Pentagrams are quantum scars resembling a classical periodic orbit which has a shape of a rounded five-pointed star, as illustrated in Figure 1. Circles are not technically perturbation scars, instead they are more related to the eigenstates of the unperturbed system. However, in this thesis circles are considered a scar for the sake of simplicity. The main classes appear with different amounts of surrounding noise. The clustering methods developed cluster all scars including images of only noise, so if there are large amounts of similar kinds of noise they will appear in the results. In Figure 2 there are examples of main classes and a possible class of noise. Due to the large amounts of scars of only noise there are also many different types of noise, but those are typically not of scientific significance. Some different types of noise are plotted in Figure 3.

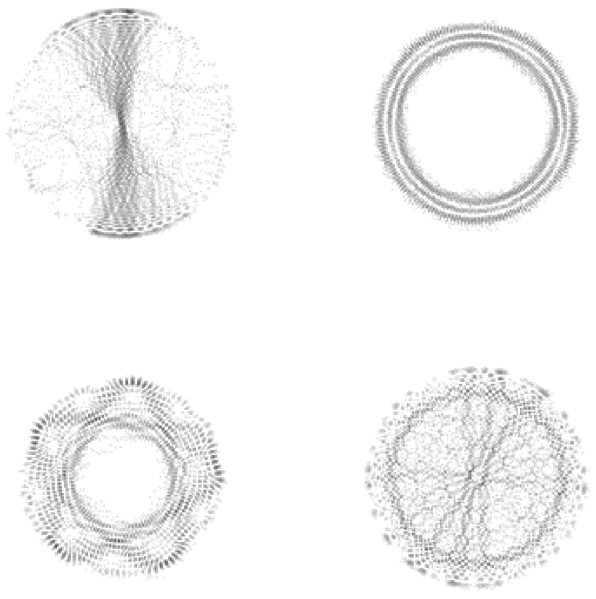

Figure 2. *Bouncing ball, circle, and pentagram -types of quantum scars and an anomaly that seems to be the combination of first two.*

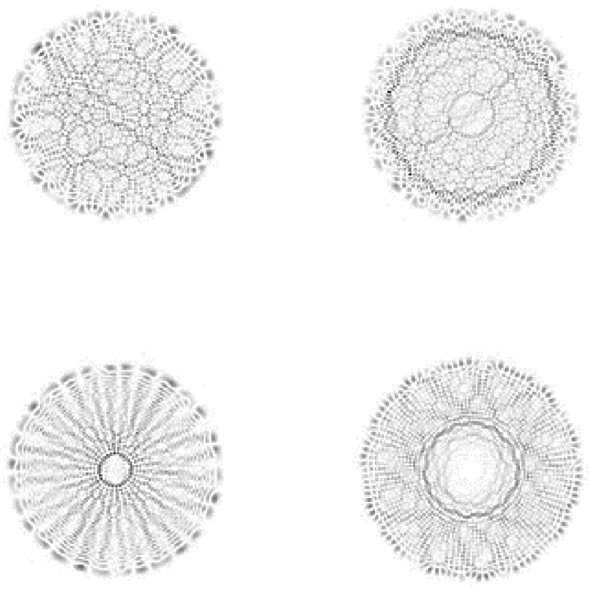

Figure 3. *Different kinds of noisy images.*

2.1 Scars in the test set

The scars of the test set are 300-by-300 matrices of complex numbers from waveform equation results. Squaring these values results in a probability space. The squared values of the waveform results in the test set range from 10^{-47} to 1.1. To regularize the software the values of any given dataset such as the test set are normalized to closed range [0,1]. Normalization is performed in two steps by first reducing the minimum of the dataset from all elements and then dividing the dataset's elements by the maximum of the dataset. Reducing 10^{-47} from all elements may seem insignificant as it does not even affect most of the double precision floating-point numbers. Regardless, the time that subtraction takes is so short that the certainty of having properly normalized data outweighs it.

Scars can be plotted for observation with the normalized matrix. By eyeing through some of the samples it can be noted that most of them are just noise. The samples also slowly grow with the index, so they are not of uniform size inside the dataset. Each class of scars in different rotations can be observed through most of the dataset, although they appear in different densities. The rotational difference between samples as well as different scale as seen in Figure 4 complicate the classification.

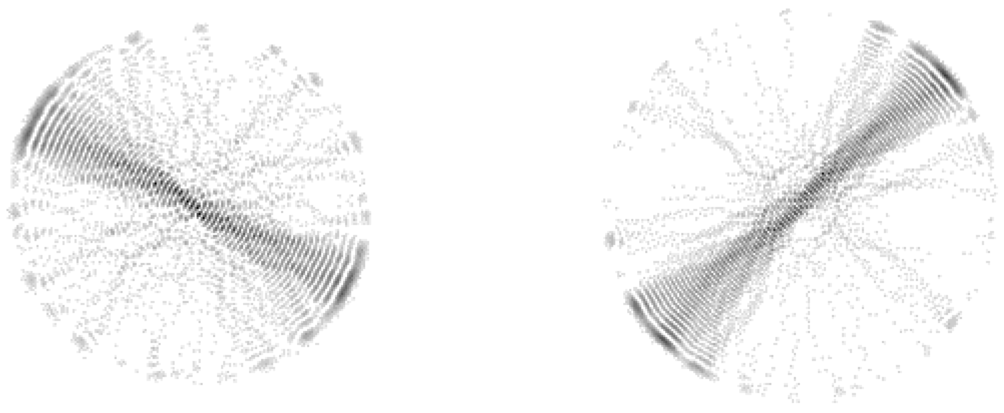

Figure 4. *Two similar scars with a different rotation and scale.*

The line between noise and actual scars in the dataset is thin, arbitrary even. Using similarity-based method for calculating distance matrixes allows users to draw the line in hierarchical clustering. This thesis does not cover where that line should precisely be drawn, merely provides the tools for it. As an example, in Figure 5 there are four scars of the same class with increasing amounts of noise.

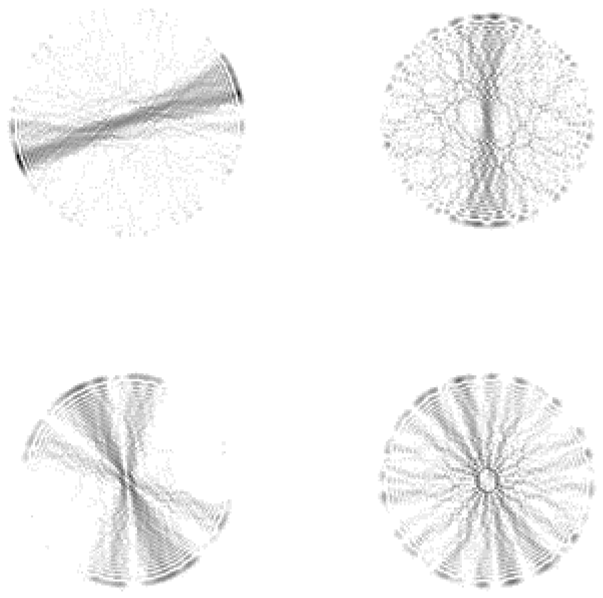

Figure 5. *Bouncing ball -type scars with increasing amounts of noise.*

Each scar in Figure 5 has a unique problem for clustering. The top-left scar is not actually symmetric, as most of the probability is located on the same, lower side of the line. The three other scars include noise spread in different ways. Top-right scar has its noise spread evenly, bottom-right has multiple weaker lines of higher probabilities and bottom-left has two stronger lines in addition to the main line. Aligning these scars computationally may not provide the intuitive alignment if another alignment with good noise overlap is found.

The scars of the test set were obtained by simulating a perturbed system similar to that of the system in [4]. From 4000 states of the lowest energy indexes 2000-4000 were extracted. The potential of the system was set as $0.5r^5$, perturbation bumps were $M = 4$ in magnitude and their full width at half maximum was 0.235. No magnetic field was applied to the system.

3. SIMILARITY OF TWO SCARS

In an ideal case with all scars in the same rotation and size the distance could be defined with just a simple subtraction between the pixel values of scars. Summing up the absolute values of element-wise subtraction gives the absolute distance between any two given scars. Absolute distance is however just one of the possible methods for measuring the distance between images. Mean squared error is by far the most used method for calculating distances although knowing all the samples are of uniform size computational time can be saved by just calculating the squared distance. Cross correlation is widely used for signals and images in signal processing and functions a bit differently than subtraction-based error estimates. [6] The common denominator for all three in cases of differing scales and rotations is that they result in total inaccuracy.

3.1 Adjusting the scale and rotation of a scar

Showing a picture in different scale is such a mundane task that there are multiple different Python libraries that offer their own solutions for it. NumPy offers resizing for the same data format the test data is in, NumPy matrixes. Python Imaging Library, PIL, also has a resizing function as does cv2-library. Each of the resizing functions has their ups and downs, but they have a mutual problem when increasing or decreasing the size of scar measurements: for an element-wise operation to take place, both matrixes need to be of same size.

The goal of resizing images of scars is to resize the scar in the middle of the picture. This can be achieved by either separating the scar from the picture and placing it back resized or resizing the entire picture. Separating the scar would require much more work than simply scaling the entire image, so in cases of both increasing and decreasing the size of the scar the resizing of entire pictures is used. After increasing the size of the entire image, it can be fitted back to wanted shape simply by removing elements other than the middle ones. Elements lost are those that were on the edge of the original picture, for scars that means values close to zero. Using this method, the scar stays focused in the middle of the picture and increases in size by the same scale the entire image was resized to. Decreasing the size of an image requires padding it to the original size for any comparison to take place. Reflecting values from the edges could be used in padding, but as explained before those values are so close to zero they do not affect the distance calculation. Therefore using NumPy.pad and padding the image back to the original size with zeroes, the size of the scars inside the pictures are resized by the same scale as the original pictures.

Rotating a picture normally also causes it to form a different shape that must be normalized back to the standard size. Rotating an image of a scar 45 degrees for example causes the image to form a diamond shape. Because the scars are round and surrounded by weak noise, simply clipping out out-of-bounds values and replacing null values with zeroes keeps the format of images uniform. Ready Python libraries that allow preservation of the original size include SciPy, PIL and cv2. These functions rotate a given image by a given value in degrees.

3.2 Additions to similarity calculation

By calculating the best scale and rotation scars can be plotted graphically for observation or a mean image can be calculated of a set of scars. For example, the second scar in Figure 3 can be rotated and scaled to match the scale and rotation of the first scar as seen in Figure 6. This requires the comparison function to store the rotation and scale used to obtain the best similarity. By giving the methods and two images in the same order to the scaralign()-function in Program 1 the images can be plotted like in Figure 6. With same rotations and scales mean images can be calculated simply by summing them together and dividing the sum by the number of images. Calculating a mean image of similar scars allows clustering results to have a graphical class base that can be used in query-based similarity calculation or classification of clusters.

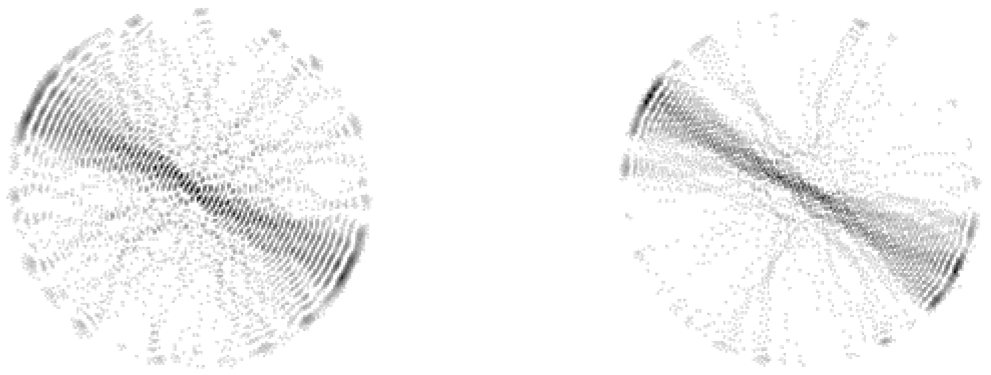

Figure 6. *Scars in Figure 4 aligned to the same rotation and scale.*

```
def scaralign(im1, im2, distance):
    #Returns copies of images with desired scale and rotation
    #Distance-input is a list [distance, rotation, size]
    #If distance[2] is negative that means scaling down the first image
    #Uncalculated distances have distance[2] = 0 by default
    image1 = np.copy(im1)
    image2 = np.copy(im2)
    if distance[2] == 300:
        pimage2 = Image.fromarray(image2)
        return image1, np.asarray(pimage2.rotate(distance[1]))
    elif distance[2] < 0:
        padsize = int((300+distance[2])/2)
        temp1 = cv2.resize(image1, dsize=(int(-distance[2]),
                                          int(-distance[2])))
        temp1 = np.pad(temp1, ((padsize, padsize),
                               (padsize, padsize)), 'constant')
        pimage2 = Image.fromarray(image2)
        return temp1, np.asarray(pimage2.rotate(distance[1]))
    elif distance[2] > 0:
        padsize = int((300-distance[2])/2)
        temp2 = cv2.resize(image2, dsize=(int(distance[2]),
                                          int(distance[2])))
        temp2 = np.pad(temp2, ((padsize, padsize),
                               (padsize, padsize)), 'constant')
        pimage2 = Image.fromarray(temp2)
        return image1, np.asarray(pimage2.rotate(distance[1]))
    else:
        raise ValueError('Distance[2] is 0')
        return image1, image2
```

Program 1. *Scaralign-function for aligning scars.*

More often than not when comparing scars, the exact best match is never found. This is either due to step size skipping over the ideal values or just because pixel accuracy does not allow it. The method of trying all rotations and sizes assures that a comparison with images at least very close to the ideal ones is calculated at some point. Applying a weak gaussian blur spreads each pixel’s value to a larger area giving pixels closer to it higher values [7]. Comparing two slightly blurred images provides close to ideal values’ distances when near the ideal scaling and rotating amounts. Blurring of images improves results with both small and large step sizes.

4. COMPARISON FUNCTION FOR SCARS

A comparison function for two scars employing the method of trying all rotations and scales should return the best score received and the rotation and scale used to get it. In terms of absolute or squared error this means the smallest found error. The opposite goes for cross correlation, where the highest sum of dot products is desired. The preservation of used method for the best similarity comes into play when calculating means of classes. The best results a comparison function can give are great scores for scars of the same type and very bad scores for scars of different type. The other variable important for the function is used time. The emphasis in the developed solution is on accuracy. Time consumption is only lowered in parts that affect the accuracy little to none.

4.1 Algorithm choices in comparison function

As discussed in chapter 3, there are multiple ready functions for rotating, scaling and calculating distances. Some ready functions are slower or simply do not work in the desired way. For example, in Table 1 there are benchmarks for 180 rotating operations. Timings were measured using timeit-library's default_timer() on an Intel i5-7400 3GHz 4-core CPU. The image rotation function by PIL is by far the fastest of the three. The quality of rotation results is equal for all functions, so PIL is used for rotation in the comparison function.

Table 1. *Benchmark results of ready rotation functions*

Rotation function	Timing includes (180 times)	Time
scipy.ndimage.interpolation.rotate()	Rotations	2.15s
cv2.rotate()	Rotations	0.16s
PIL.Image.rotate()	Conversion to Image, rotations, conversion to NumPy matrix	0.05s

Choosing between different resizing functions is simple as the only one that works consistently when scaling down is the one by cv2. Scaling down both images one after another is preferred over scaling one image up and down. This is due to some scars being so big that using the scale from smallest scar to largest on larger scars causes parts of larger scars to be cropped out. Any distances obtained with partial images are not valid, and while there is little evidence in the test set that these distances could be the best of all rotations and scales, the possibility and thus threat exists. For images where no cropping out of relevant information happens the results are identical for both methods. The only downside of scaling down both images is that any calculated mean images of pairs of images are always the size of the smaller one. This complicates calculating means of larger amounts of scars. Even the smallest scars are still quite easily distinguishable, so this does not cause any issues in results.

Selection of the right distance calculation function has less emphasis on speed and more on the difference of results for same-class scars compared to noise. Cross correlation provides the best results for very similar scars but falls short when comparing e.g. bouncing-ball-scars of different widths. Due to the normalization to range [0,1] two pixels with zero as values do not increase the similarity score and pixels with values zero and one, the maximum distance for pixels, do not subtract from it: only aligning pixels with high values add to the score. Testing cross correlation reveals better similarity scores towards evenly spaced high magnitude noise than some scars of the same class. Cross correlation is thus not a valid option for distance calculation in value range zero to one. However, special normalization methods like NCC and ZNCC can be used to modify the range to include negative numbers [6]. Pixels with low values then add to the similarity score and pixels with high and low values subtract from it. Different normalizations not only complicate the scaling and turning of images but suffer from the same problem of scars yielding better results towards high magnitude noise. Subtraction-based similarity estimates are more suited to comparing scars.

The other two distance calculation functions, absolute distance and squared distance, provide very similar results. Their difference resides in the weight of penalties for high and low subtraction results. Both methods leave a threshold for score that separates scars of the same class from noise. The difference grows to be significant when using blurred images. Gaussian blurring is added to the scars in the comparison function because the pixels may not align with any rotations. Comparing blurred images contains more lower distances for values, so penalizing lower distances less and higher distances more using squared distance provides the largest margins between scars and noise. All three distance calculation methods are within a 10% margin in time used.

4.2 Step sizes, ranges and calculation order

The time used by the comparison function is inversely proportional to the step sizes used in resizing and rotating. Doubling the step size in either rotation or resizing cuts the amount of calculations and time used in half. The effect in accuracy however is not linear. The lowest step size in rotation is determined by the bouncing ball -scars. The similarity for two bouncingball-scars is lowest for a period of 2-3 degrees, where the similarity score varies very little. A step size of three degrees results in scar-to-noise -scores for some comparisons, so a fully certain step size of two degrees is preferred. The minimum step size for resizing is set by circle-scars instead. The step size must be an even number to maintain the centering of scars. The similarity scores of circle-scars with step sizes two and four are almost identical, but with a step size of six some comparisons result in scar-to-noise -scores. Thus, a step size of four is selected.

Due to the classical orbits in scars perfect aligning can be achieved with less than 180-degree rotations clockwise. Circles do not require rotating and pentagrams can be aligned with less than 72-degree rotations clockwise but bouncing ball -scars require rotations up to half a circle. Other types of quantum scars not present in the test set such as clovers also require less than 180-degree rotations clockwise.

The range for resizing values is determined by the smallest and largest scars. Within the test set the diameter of relevant scars varies from a bit under 200 pixels to about 240 pixels. 240 pixels wide scars in 300-by-300 matrixes need to be resized to 250-by-250 matrixes for scars to have diameters of 200 pixels. To be certain that the largest required rescaling is calculated for future datasets both images are resized in a range from 240 to 300.

Calculation order does not affect the results of the comparison function in any way, but it does affect the time consumed. The only calculation order that matters is whether to calculate all rotations for scales or all scales for rotations. By using the forementioned step sizes and ranges two versions of the function can be timed. Timings of both versions reveal that calculating all scales for rotations consumes 10% less time. This method is thus preferred.

4.3 The execution of the comparison function

The comparison function in Appendix A takes two images of scars as input and returns a list that contains the lowest distance found and the scale and rotation used to obtain it. The first action in the function is copying both images so the reference semantics in Python do not change the input values. A weak gaussian blur is added using cv2.GaussianBlur with a 5-by-5 filter and out-of-range values assumed as zeroes. The output value is defined with a distance value greater than any normalized scars' comparison can provide to ensure its overwrite.

Using the PIL rotation function requires changing the matrix to rotate into Image. The transformation to Image and back to a NumPy matrix is not safe but can be considered as a linear function. The transformation is applied to both blurred images to ensure the identical distances for calculating distances between scars A and B and between B and A. The image to be rotated is kept in the Image-format to save time inside the loops while the other image is instantly transformed back to a NumPy matrix.

The outer loop in the comparison function consists of rotating the Image-format image and a single comparison without any resizing. The Image-format image is transformed back to a NumPy matrix before any comparison takes place. For each comparison the sum of squared distances between pixels is compared to the lowest distance yet obtained. If a lower distance is found that distance, used rotation, and size are stored to the output value.

Within the outer loop there are two inner loops containing downscaling operations and for both images. Either the unrotated or the rotated image is resized into a new size determined by “newsize”. The required padding width is calculated for each resizing and applied to resized images padding them back to original size. The comparisons using the resized unrotated image store the size-variable in output as negative. Without this there would be duplicate methods and aligning the scars based on the output value would not be possible.

4.4 Distances for different classes of scars

The comparison function returns very consistent distances for clear class examples. Bouncing ball -scars for example yield distances with a scalar value of 40-70 towards every other type of scars and distances of less than 20 towards other bouncing ball -scars. Circle-scars alike yield high distances 30-70 towards other types of scars and distances of 20 or less towards other circle-scars. Pentagrams however are much closer to noise as they yield distances around 17-28 towards noise, 40-50 towards bouncing balls and circles and 5-15 towards other pentagrams.

Not all of the scars are clear representatives of their classes. With increasing amounts of noise all types of scars yield shorter distances towards noise and longer distances towards their respective classes. There are many noisy class examples that have shorter distances to some noisy images than clear class examples. However, the nature of their distances remains. Noisy bouncing ball -scars yield longer distances toward circle scars and lower distances towards bouncing balls than just noise would. Recognizing this aids in the search for a distance metric for hierarchical clustering.

4.5 Calculative load of the comparison function

The comparison function even when optimized has a large calculative load. With the step sizes presented the squared error and sum is calculated for the 90 000 matrix elements 2790 times. Computing with the same Intel i5-7400 3GHz 4-core CPU as in benchmarking the comparison of two images takes on average 1.2 seconds of time. This means calculating distances from a single scar to every scar in the 2000 image test set takes 40 minutes and calculating every possible distance takes over 55 days.

The comparison function is safe for the dataset so parallel computing can be fully implemented. Threading the distance calculation may speed up calculation if there are multiple cores available, or alternatively the distances to be calculated can be evenly spread among multiple computers as the distances are not dependent on each other. The simplest of solutions for getting results in a reasonable amount of time is employing a supercomputer to their calculation.

5. HIERARCHICAL CLUSTERING FOR SCARS

Hierarchical clustering uses a distance matrix to cluster elements sequentially until only one cluster is left. A distance matrix contains distances from every element to every element, and by definition the distance from an element to itself is zero. The matrix is symmetric, so the distance between two elements is the same for both ways. [8] The used function for clustering, scipy.hierarchy.cluster.linkage(), requires a full distance matrix, but not all of the matrix need to be calculated. When calculating a distance matrix for scars each distance is mirrored and distances to self are by default zero. Mirrored methods for obtaining the lowest distances are not valid, but they are not used either.

5.1 Hierarchical clustering methods

Hierarchical clustering consists of three key variables: the distance matrix, clustering method and distance metric. The clustering is a recursive function that either joins two elements to form a cluster, adds an element to an existing cluster or combines two clusters. Clustering finishes after all elements are inside one single cluster. Intermediate clustering results can be picked based on the number of remaining clusters and single elements or a threshold that does not allow combining clusters with larger than a set distance.

The comparison function gives scalar values for the distance matrix, but different distance metrics can be used to interpret them. Euclidean distance for example means considering each of them to exist in n-dimensional space with the distance matrix's distances between them. The clustering function has over a dozen existing distance metrics and one can also use their own. Due to the nature of distances for different classes of scars explained in chapter 4.4, correlative distance works exceedingly well [9]. While similar scars may have a large distance, their distances to their own class and other scars correlates highly.

There are multiple clustering methods available on the clustering function. Some are very simple concepts, like Nearest Point Algorithm, where the shortest distance is always joined be it between clusters or elements. Other methods include UPGMA and WPGMA, where each cluster is assigned a single center value, and WARD that utilizes the Ward variance minimization algorithm. [10] Using correlative distance as a metric results in only UPGMA and WPGMA clustering correctly. The results of UPGMA and WPGMA are almost identical, but their difference resides in the threshold margins for correctly clustered classes, which are a little wider for UPGMA. Therefore, UPGMA is used for clustering.

5.2 Dendrogram and clustering behavior

Hierarchical clustering sequentially combines elements until only one cluster is left. This means that the end result of clustering is trivial and of no significance. The process of clustering however is very relevant and key to setting thresholds where clustering should be stopped. The common way of graphically representing the process is using a dendrogram, which vertically shows the distance values where elements or clusters were combined [11]. A dendrogram is shown in Figure 7 using UPGMA and correlative distance for a selected set of 200 scars spaced 7 images apart starting at state 2000 (states 2000 + 7n).

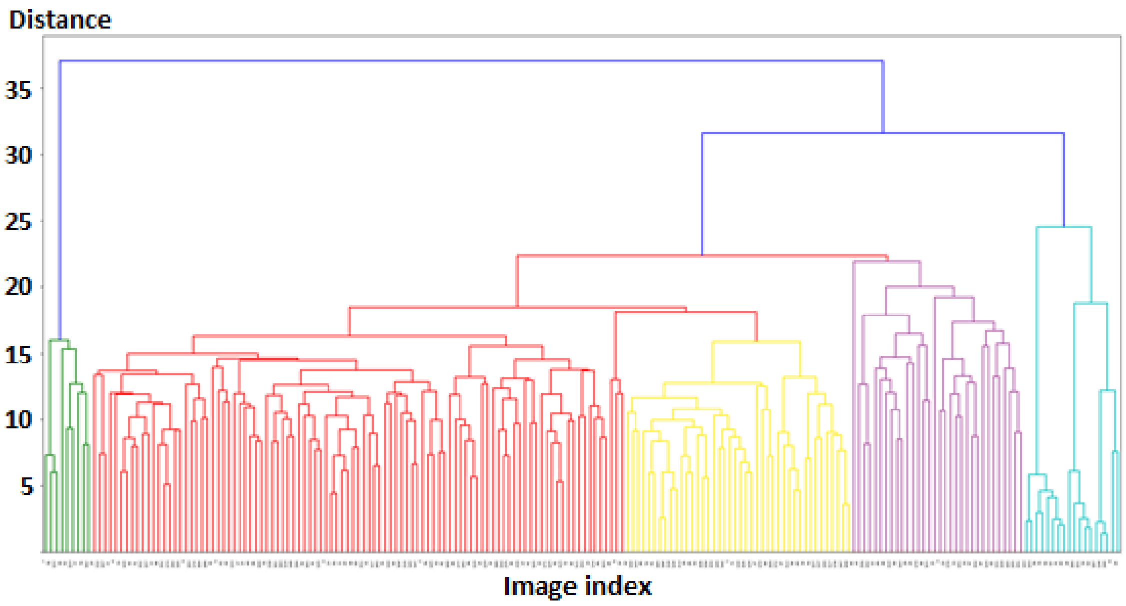

Figure 7. *Dendrogram of a 200-scar set.*

The colored clusters in Figure 7 represent clear bouncing balls in green, circles in blue, pentagrams in yellow, noisy bouncing balls in purple and noise in red. In the 200-image set noisy bouncing balls are closer to noise than they are to clear bouncing balls, but this may not always be the case. If a single threshold for clustering had to be set based on the dendrogram, the earliest class to be clustered out is pentagram-scars. The pentagram-class is accurately clustered between distances 16-18 after which it is combined with noise.

In Appendix B there are all individual images from each cluster except noise. The bouncing balls in green are good class examples as are all circles in blue. Pentagram-scars include class examples and some scars that are harder to distinguish. The cluster of noisy bouncing balls consists of every other scar in the selected set that includes a strong enough line. The earliest class to be clustered out is pentagrams, so setting a threshold that prevents it causes circle-scars to form two or three clusters and noisy bouncing balls to form several. The noisy bouncing balls -cluster has subclasses that are extracted when using a threshold for pentagrams, so the entirety of the cluster is never seen as a whole.

5.3 Graphical representation of clustering results

While the dendrogram is an informative way of presenting clustering results, classification is impossible from it alone. The clustered classes could appear in any order and in different sizes, not to mention the classes appearing in a given dataset might also be unknown. Classifying from the dendrogram requires some sort of graphical representation for each cluster. The scaralign()-function from Program 1 can be employed to align the scars of a cluster, enabling the forming of mean images for each cluster.

To get the images for each cluster a simple transformation of the clustering must first be stopped at a given point. A ready function, scipy.cluster.hierarchy.fcluster(), does this for either a set number of last remaining clusters or a threshold for distance. Clustering results are provided in an array with a length of the number of clustered images containing the index number of each image's respective cluster. The indexes of images inside the dataset need to be extracted from these results. Program 2 unravels the clustering results and passes clusters to mean image calculation.

```
def meanimages(clustering, distancematrix, data):
    numclasses = clustering.max()
    images = []
    for i in range(1, numclasses+1):
        indexes = []
        for j in range(0, clustering.size):
            if clustering[j] == i:
                indexes.append(j)
        #Note that at this point indexes are in rising order
        images.append(meanimage(indexes, distancematrix, data))

    return images
```

Program 2. *Gathering image indexes of clusters, loop for mean images.*

Inputs for Program 2 are the vector from fcluster(), calculated distance matrix and data structure for images. Distance matrix and data structure are not used within the function, only passed on to mean image calculation. The outer loop loops through all clusters in the clustering results and after receiving the image indexes calls a mean image calculation on those images. Within the inner loop all elements of the clustering result are looped through storing the indexes of images belonging to the cluster determined by the outer loop. Because the looping is executed in rising order, the stored indexes are also in rising order. Thus, methods used to align scars to other scars later in the list are always valid.

For the mean calculation itself distance matrix is only provided to reduce redundant calculating. In theory no new comparisons are needed to form mean images, as the best alignment methods are already calculated. If comparison would only scale one image, the task would be as trivial as to align every scar to the first one. Because the comparison could result in the need to scale either image, a more complex approach is required. The rotation needed for alignment is not dependent on scaling, so every scar can be rotated to the best rotation towards the first scar defined in the distance matrix. Because images are only scaled down, it is logical to presume the size of the aligned scars to be the same as the smaller scar. Summing the aligned scars together should create a sum of aligned scars the size of the smallest scar within the summed images. Aligning more scars into this sum can be achieved by scaling with the scaling factor found in the distance matrix between the smallest scar of the sum and the scar to be aligned. Dividing the sum by the number of images provides the mean image.

The full redundancy of mean images only works in theory, as a small but significant number of alignments are not intuitive. In sequential calculation one imperfect scaling ruins the entire sequence afterwards as the alignments are not associative. To improve accuracy, the distance matrix results can be applied to smaller groups of images resulting in multiple summed images that can be aligned using the comparison function. In Program 3 there is the mean image calculation called in Program 2 employing a groups size of two.

```
def meanimage(indexlist, distancematrix, data):
    numimages = len(indexlist)
    if numimages == 1:
        return np.copy(data[:,:,indexlist[0]])
    tempmeans = []
    for i in range(0,numimages-1, 2):
        temp1, temp2 = scaralign(data[:,:,indexlist[i]],
                                 data[:,:,indexlist[i+1]],
                                 distancematrix[i,i+1,:])
        tempmeans.append((temp1+temp2)/2)
    parity = 1
    if numimages % 2 == 1:
        tempmeans.append(data[:,:,indexlist[-1]])
        parity = 0
    numtemps = len(tempmeans)
    if numtemps == 1:
        return tempmeans[0]
    mean = tempmeans[0]
    if numtemps > 2:
        for i in range(1,numtemps-1):
            distance = compare(mean, tempmeans[i])
            mean, tempmeans[i] = scaralign(mean,
                                          tempmeans[i], distance)
            mean = ((mean*i) + tempmeans[i])/(i+1)
    distance = compare(mean, tempmeans[-1])
    mean, tempmeans[-1] = scaralign(mean, tempmeans[-1], distance)
    #If last tempmeans item is a single image and not a mean
    if parity == 0:
        #Calculate mean but weigh the image by half of normal
        mean = ((mean*(numtemps-1)*2) + tempmeans[-1])/((numtemps*2)-1)
    #If last tempmeans image is a mean image weigh normally
    else:
        mean = ((mean*(numtemps-1)) + tempmeans[-1])/(numtemps)

    return mean
```

Program 3. *Mean image calculation.*

6. RESULTS OF HIERARCHICAL CLUSTERING

The classes to be clustered are clear and noisy bouncing balls, circles and pentagrams. If only clear bouncing balls and circles need to be clustered, the last three clusters can be selected from clustering. In Figure 8 there are mean images calculated with Programs 2 and 3 for the last three clusters for three 200-image sets. These three images contain images of clear bouncing balls, circles and the rest. Due to the different amounts of circles in circle-scars, the mean image is not very representative, but still quite distinguishable. The classes appear in random order, so classification by hand is still required. The dendrograms of these three new subsets are presented in Appendix C in the same order.

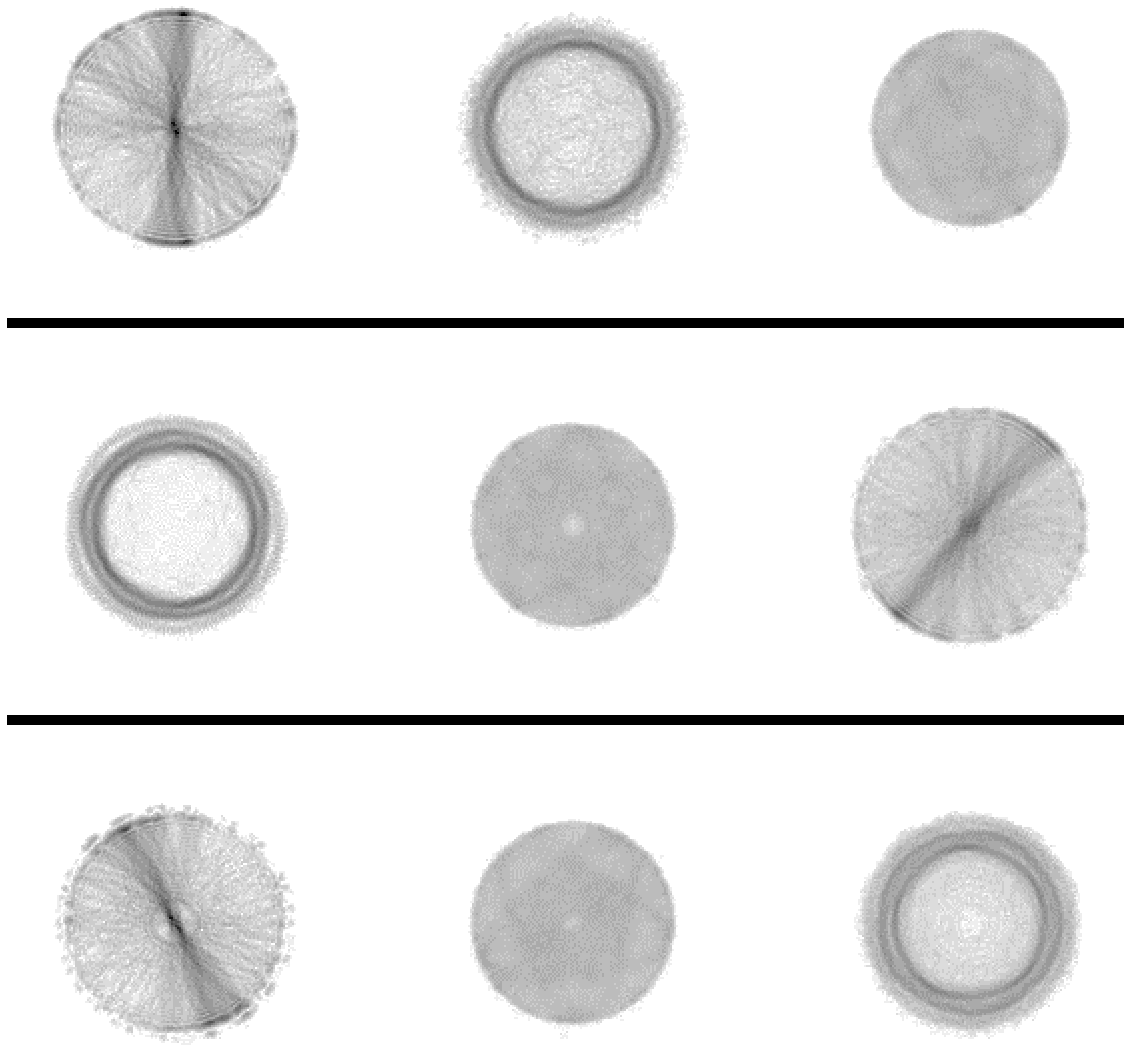

Figure 8. *Mean images of last three clusters for three 200-image sets.*

The other important clusters were noisy bouncing balls and pentagrams. The threshold value for a single pentagram-cluster was 16-18, while noisy bouncing balls were combined with higher distances. Setting the threshold to 17 provides 9-12 clusters. In Figure 9 there are mean images of pentagram-clusters with distance threshold of 17 for the same 200-image sets as in Figure 8. The resulting multiple clusters of noisy bouncing balls for the same three sets are plotted in Figure 10.

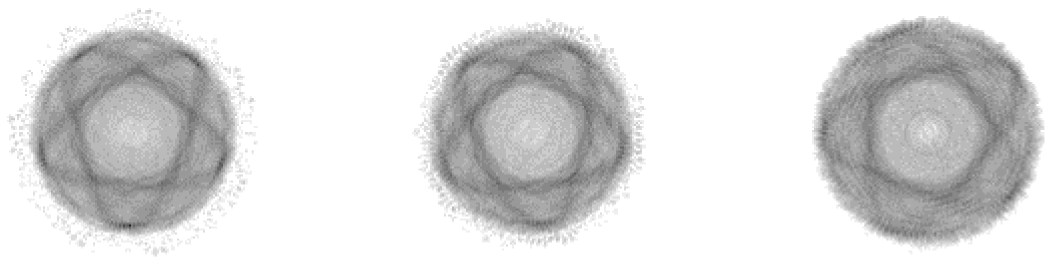

Figure 9. *Mean images of pentagram-clusters from three 200-image sets.*

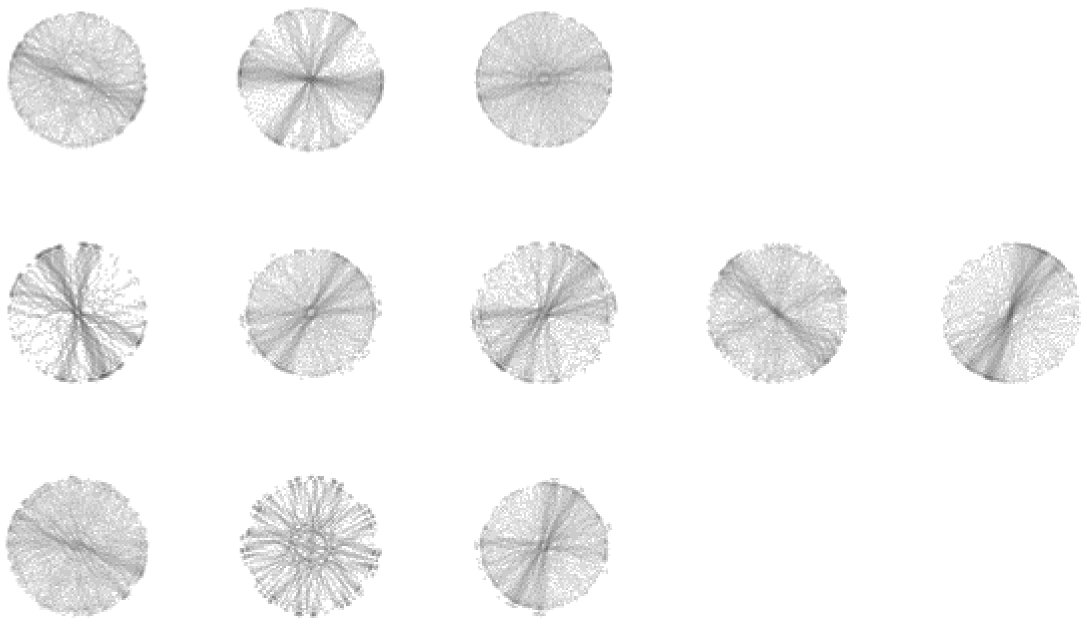

Figure 10. *Mean images of noisy bouncing ball -clusters from the three sets using a threshold for pentagrams.*

7. CONCLUSIONS

The goal of this thesis was to provide means for accurate clustering of perturbation induced quantum scars. These means include a comparison function, distance matrix forming, hierarchical clustering, scar aligning and mean image calculation for presenting clustering results. All of these can be executed using the Python code and functions developed during this thesis.

The software built for this thesis is capable of clustering quantum scars for at least the used data set. To preserve all desired classes a threshold must be used. Based on smaller subsets of the data set this threshold would be 17, leading to around 10 clusters that must be classified by hand. This threshold and the software itself are not yet tested on larger sets of images, so the best threshold value may differ. With a larger number of scars, the test set may also cluster noisy bouncing balls and clear bouncing balls together. The clustering method, UPGMA, should keep noise as distant from clustered classes as in the smaller image sets.

The built software is generalized for any type of classical orbits, so it should work for new classes of scars as well. Any required thresholds can be tested out with help from a dendrogram and mean image calculation. If non-circular boundaries for quantum scars are used, the comparison function provides accurate results only if the boundaries are removed from images. For three-dimensional systems, the comparison function can be modified to include a 360-degree turn range over the new axis. However, the calculative load for three-dimensional images is so heavy that required computational resources may be too expensive.

REFERENCES

[1] C Wolfgang P. Schleich, Daniel M. Greenberger, Donald H. Kobe and Marlan O. Scully, Schrödinger equation revisited, PNAS April 2, 2013. 110 (14) 5374-5379

[2] L. Kaplan, Scars in quantum chaotic wavefunctions, Nonlinearity, 12 (1999), pp. R1–R40

[3] E. J. Heller, Phys. Rev. Lett. 53, 1515-1518 (1984)

[4] P. J. J. Luukko, B. Drury, A. Klales, L. Kaplan, E. J. Heller, and E. Räsänen, Sci. Rep. 6, 37656 (2016).

[5] J. Keski-Rahkonen, P. J. J. Luukko, L. Kaplan, E. J. Heller, E. Räsänen, Phys. Rev. B 96, 094204 (2017).

[6] Sonam Singh, Dinesh Ganotra, Modifications in Normalized Cross Correlation Expression for Template Matching Applications. Available: https://pdfs.semanticscholar.org/5f3b/48c8526a056c80e13b42aaa2b1287b6d2790.pdf

[7] R. Fisher, S. Perkins, A. Walker and E. Wolfart, Spatial filters – Gaussian smoothing, 2013, https://homeages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[8] Tim Bock, What is a Distance Matrix, Displayr, https://www.displayr.com/what-is-dendrogram (referenced 6.5.2018)

[9] The SciPy Community, correlative distance, https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.distance.correlation.html (referenced 4.11.2018)

[10] Daniel Müllner, Modern hierarchical, agglomerative clustering algorithms, 2011, Available: https://arxiv.org/abs/1109.2378v1

[11] Tim Bock, What is a Dendrogram, Displayr, https://www.displayr.com/what-is-dendrogram (referenced 6.5.2018)

APPENDIX A: THE COMPARISON FUNCTION

```
def compare(Image1, Image2):
    copy1 = np.copy(Image1)
    copy2 = np.copy(Image2)
    #Apply gaussian blur for less dependency on pixel perfect fitting
    blur1 = cv2.GaussianBlur(copy1, (5,5), 0)
    blur2 = cv2.GaussianBlur(copy2, (5,5), 0)
    #90001 is overwritten if any comparison is made.
    bestmatch = [90001,0,300] #Distance, angle, size
    prep = Image.fromarray(blur1)
    image1 = np.asarray(prep)
    image2 = Image.fromarray(blur2)
    #Rotation up to half a circle with a step size of two degrees
    for i in range (0,180,2):
        rotated = image2.rotate(i)
        rotated = np.asarray(rotated)
        tempsum = np.sum(np.sum(np.square(np.subtract
                                          (image1, rotated))))
        if tempsum < bestmatch[0]: #for scale 1:1
            bestmatch = [tempsum, i, 300]
        #All newsizes are even for keeping the centering of images
        for newsize in range(240,300,4): #for scales < 1
            padsize = int((300-newsize)/2)
            temp = cv2.resize(rotated, dsize=(newsize, newsize))
            temp = np.pad(temp, ((padsize, padsize),
                                 (padsize, padsize)), 'constant')
            tempsum = np.sum(np.sum(np.square(np.subtract
                                              (image1, temp))))
            if tempsum < bestmatch[0]:
                bestmatch = [tempsum, i, newsize]
        for newsize in range(240,300, 4): #for scales > 1
            padsize = int((300-newsize)/2)
            temp1 = cv2.resize(image1, dsize=(newsize, newsize))
            temp1 = np.pad(temp1, ((padsize, padsize),
                                   (padsize, padsize)), 'constant')
            tempsum = np.sum(np.sum(np.square(np.subtract
                                              (temp1, rotated))))
            if tempsum < bestmatch[0]:
                #Newsize negative for 1st image resizes for keeping note
                bestmatch = [tempsum, i, -newsize]
    return bestmatch
```

Program 4. *The comparison function.*

APPENDIX B: CLUSTERED INDIVIDUAL IMAGES

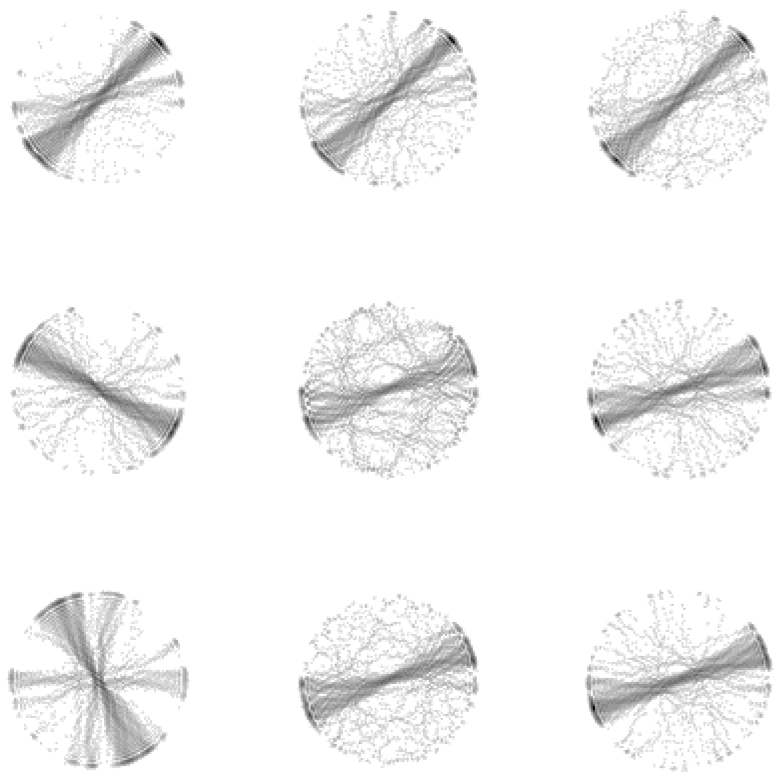

Figure 11. *Clustered individual clear bouncing ball -scars.*

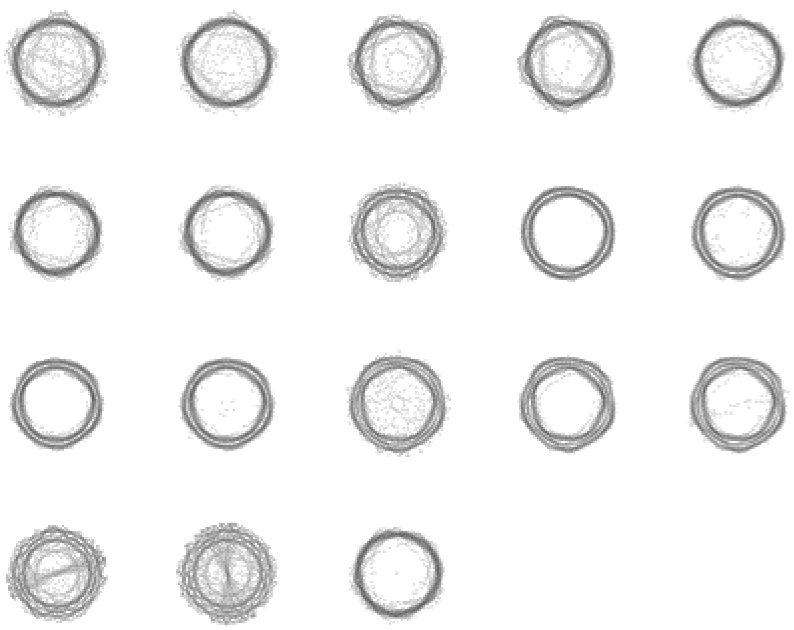

Figure 12. *Clustered individual circle-scars.*

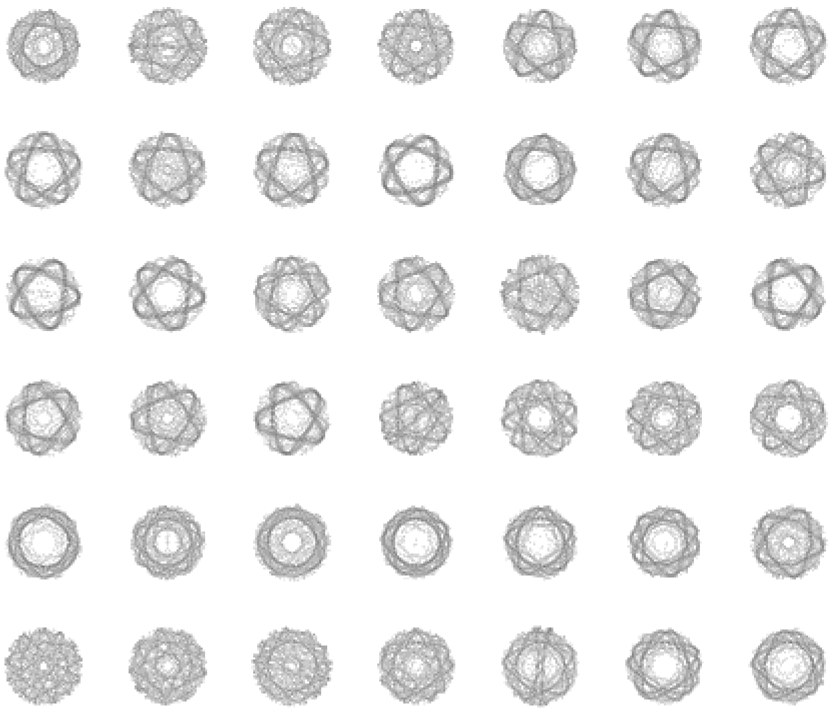

Figure 13. *Clustered individual pentagram-scars.*

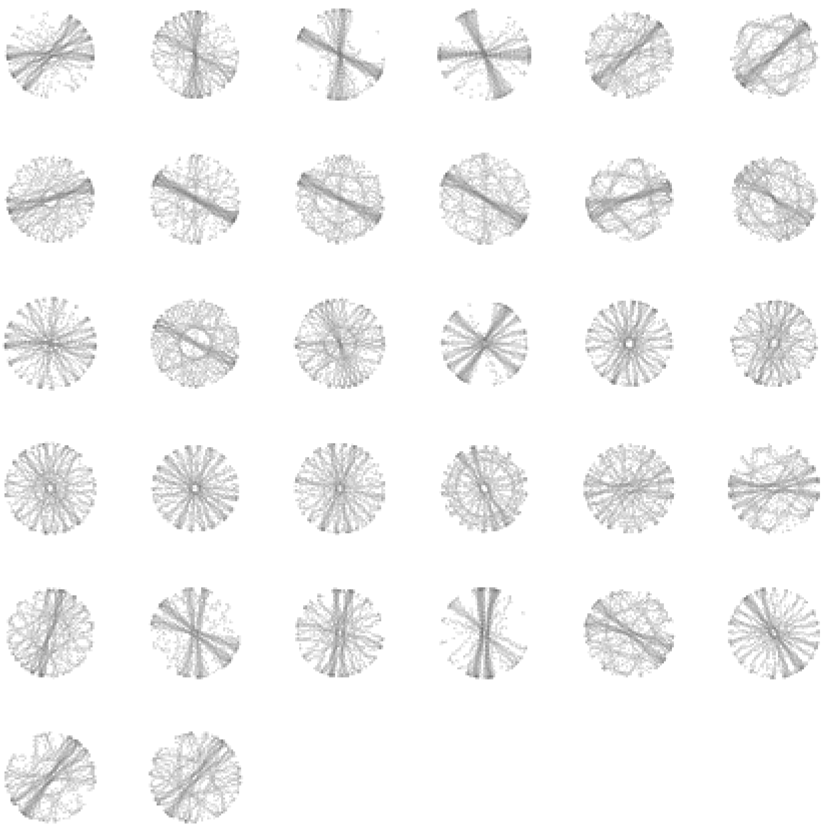

Figure 14. *The cluster of noisy bouncing balls. With a threshold for pentagrams this is divided into multiple clusters.*

APPENDIX C: ADDITIONAL DENDROGRAMS

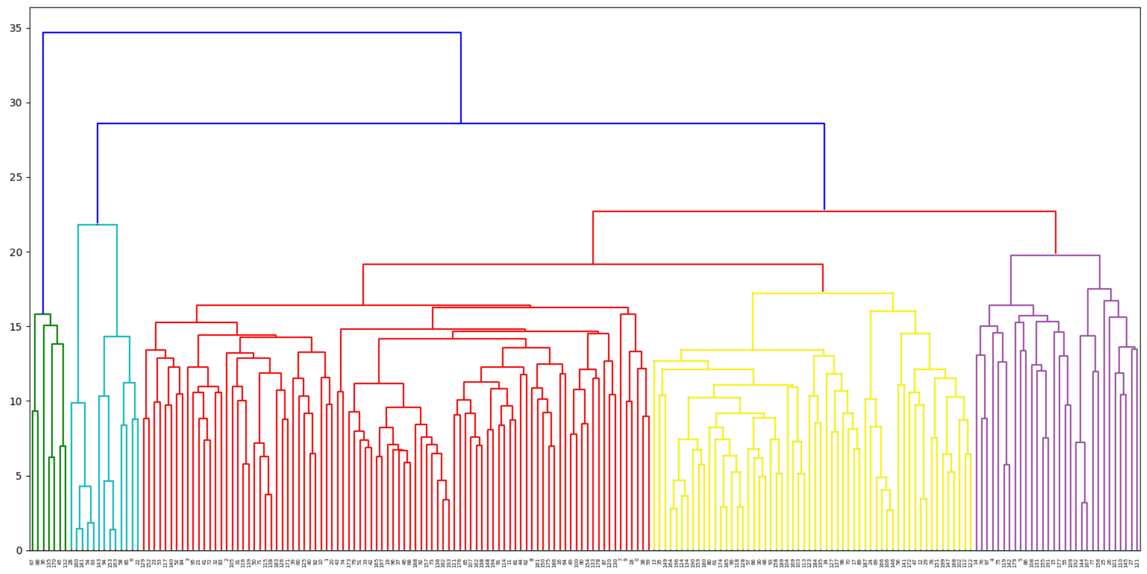

Figure 15. *Dendrogram for 200 states of form 2001+7n.*

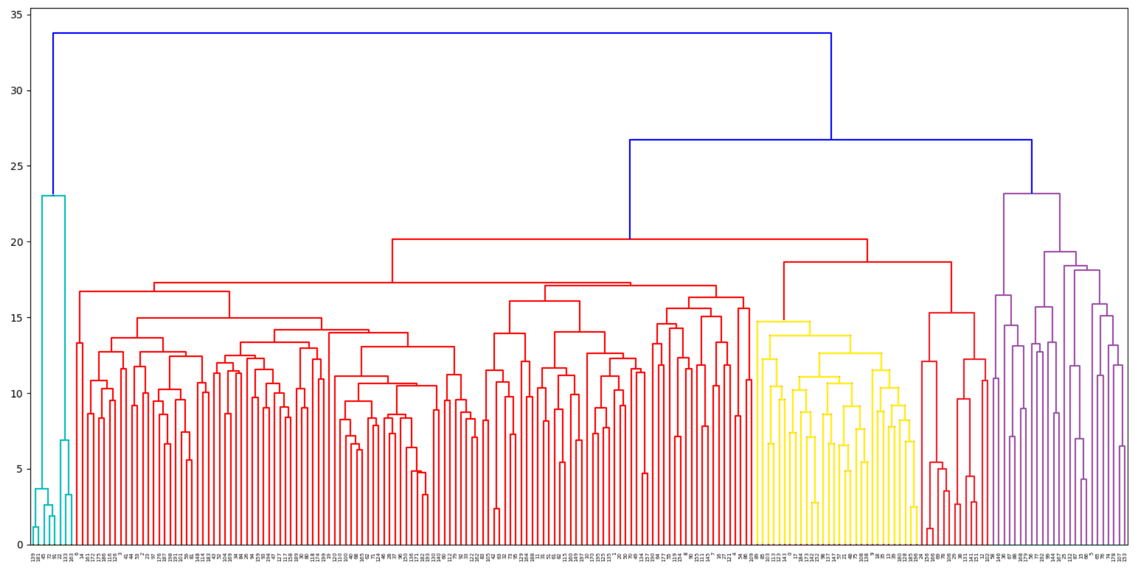

Figure 16. *Dendrogram for 200 states of form 2002 +7n.*

In Figure 16 there is no cluster of clear bouncing balls as they are clustered together with noisy ones. Also, between the yellow and the purple clusters there’s an anomaly where multiple quadruple circles appear. This gives certainty to the system’s capability to spot new classes and anomalies.

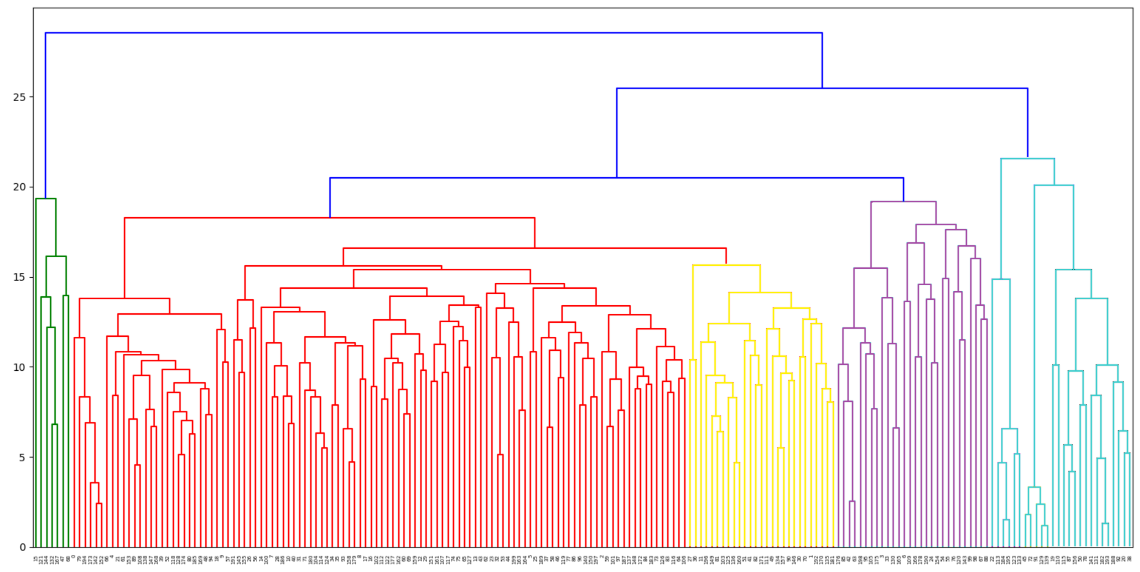
Figure 17. *Dendrogram for 200 states of form 2003 +7n.*