Short Stories on Data Science

Collected from the blog posts at temdm.com

Pavel Potapov

with contribution of

James Bond

Contents

1 How much noise can we remove by PCA?

1.1 Getting acquainted with Mr. Bond

Y'all probably heard about Principal Component Analysis (PCA) and how it can be used to clean up noisy datasets. This can be done with our software, for instance. But have you ever wondered how it actually works? And more importantly, can it eliminate all the noise or just a fraction? Well, this post is here to shed some light on those questions. Let's dive in!

Figure 1: Secret agent James Bond.

Let's break down the concept of Principal Component Analysis (PCA) in simple terms. Imagine we have a spy named Mr. Bond, who's tasked with sending reports from a top-secret school. These reports contain student grades in different subjects. Each week, Mr. Bond creates a table (shown in blue in the figure) where each row represents a student, and each column represents their scores in specific subjects like math, sport, and geography. Now, here's the catch: Mr. Bond can't transmit the table as is because it needs to be encoded to maintain secrecy. To do this, he has a set of predefined keys (shown in yellow). He simply multiplies the blue matrix (the grade table) with the yellow matrix (the keys) to obtain a new matrix shown in green. This encoded matrix is then transmitted via radio during the cover of night.

At the headquarters, the smart folks there know linear algebra. They receive the encoded matrix and use the reverse key matrix to decipher it and restore the original table with the students' grades.

After a few weeks, Mr. Bond starts to notice a peculiar pattern with the keys he receives from the headquarters. It turns out that these keys aren't just random combinations. Whenever he multiplies any column of the key matrix with another column, the result is always zero. It dawns on him that his lazy boss, who designed the keys, took a rather simplistic approach.

You see, his boss considered the three subjects as coordinates in a 3D space, and all he did was rotate these coordinates to mix up the results. Each week, he came up with three new basic vectors and defined them in terms of the original coordinates, as shown in the figure. In his old-fashioned ways, the boss made sure that the basic vectors were always orthogonal to each other. That's why multiplying the coordinates

Figure 2: Rotation of data matrix.

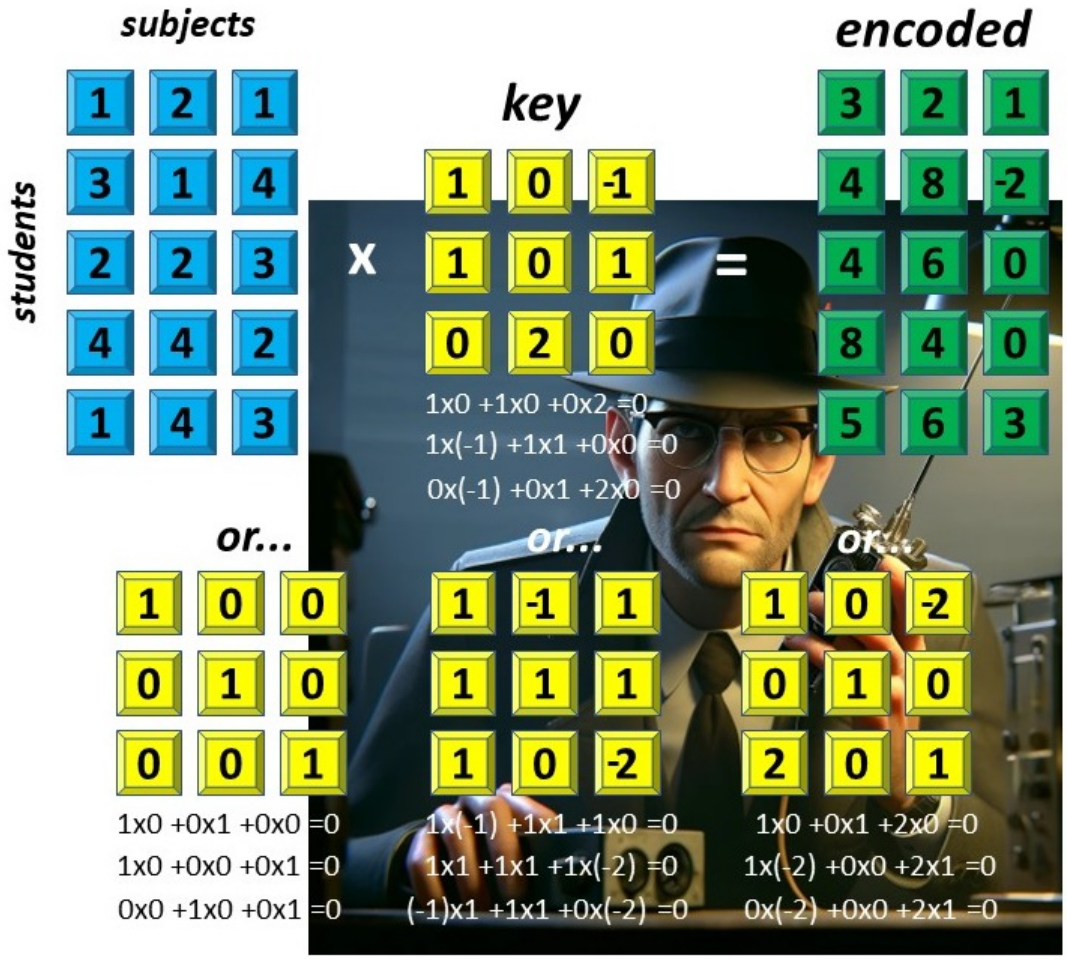
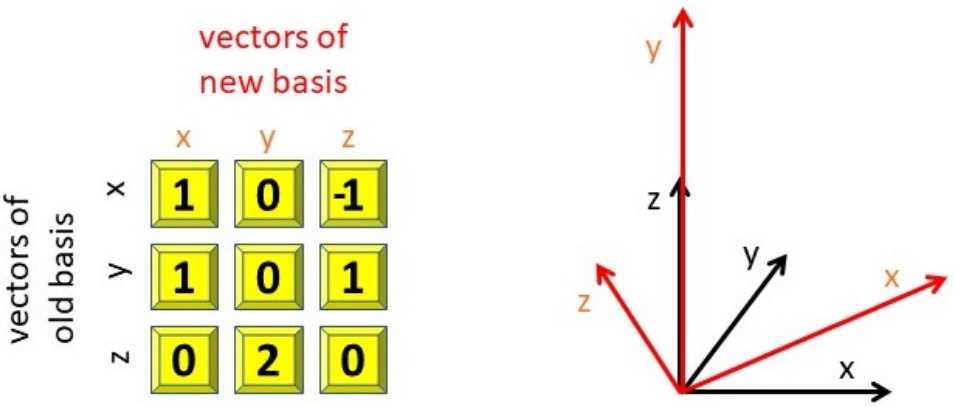


Figure 3: Rotation basis.

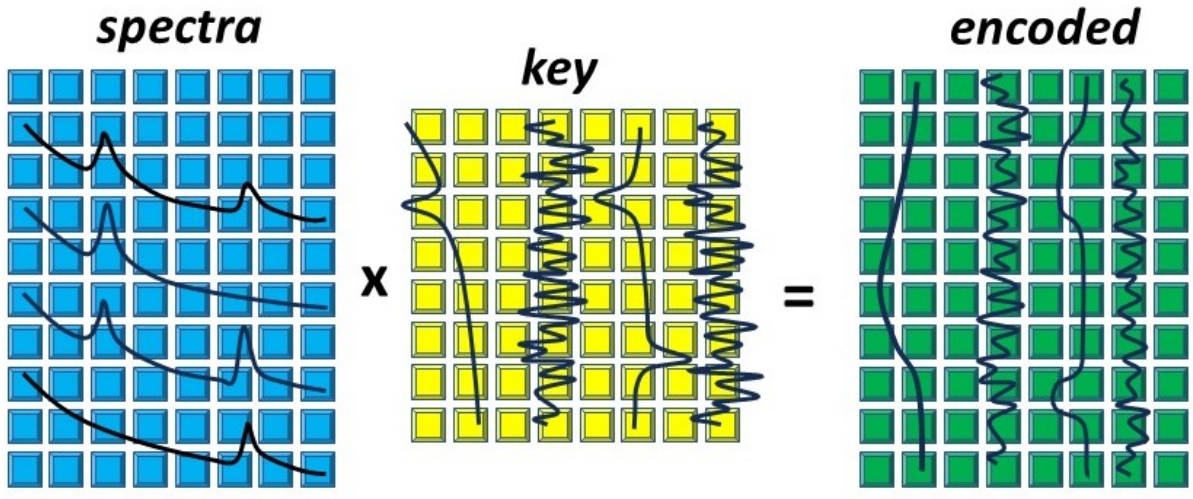


of these basic vectors always yields zero. Now, here's the funny part: For one week, his boss provided a unit matrix (the most left matrix in the first figure). This meant that there was no encoding that week!

As the story unfolds, Mr. Bond is now tasked with transmitting a number of spectra obtained from a highly classified material. Each spectrum consists of 2048 channels, which means Mr. Bond receives

2048x2048 key matrices for encoding. Just like before, his boss continues to construct the encoding matrices by rotating the basis vectors, but this time operating in a vast 2048-dimensional space.

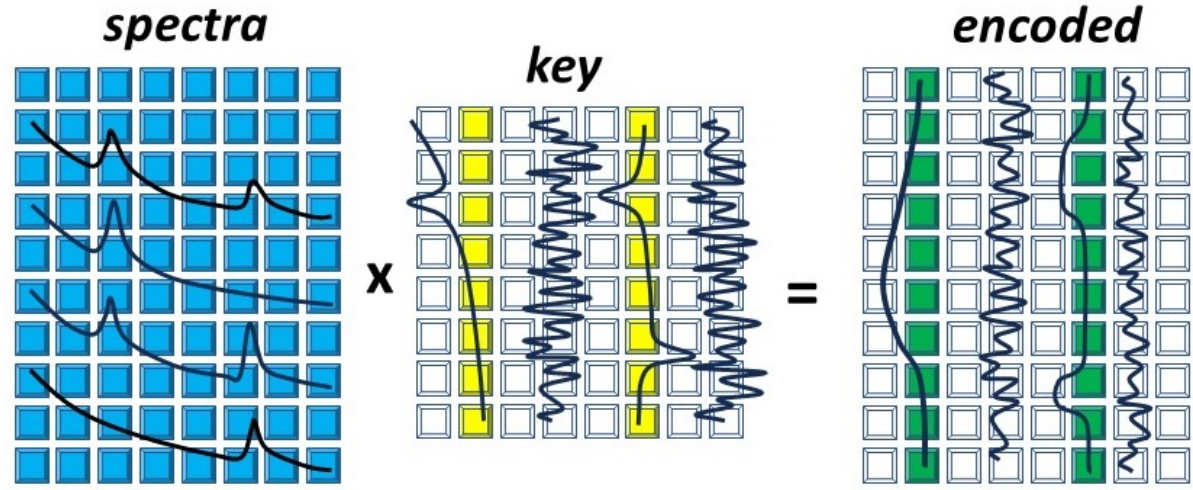
Figure 4: Now data are spectra.



As Mr. Bond faithfully transmits the encoded spectra, he begins to notice something interesting. Certain columns in the key matrices seem to be more efficient at encoding than others. These columns, presumably manually defined by his boss, result in smooth variations when applied to the spectra. However, there are other columns (probably appearing due to the orthogonality constraints) that produce noisy columns in the encoded matrices. Mr. Bond becomes skeptical about whether these noisy columns carry any valuable information at all.

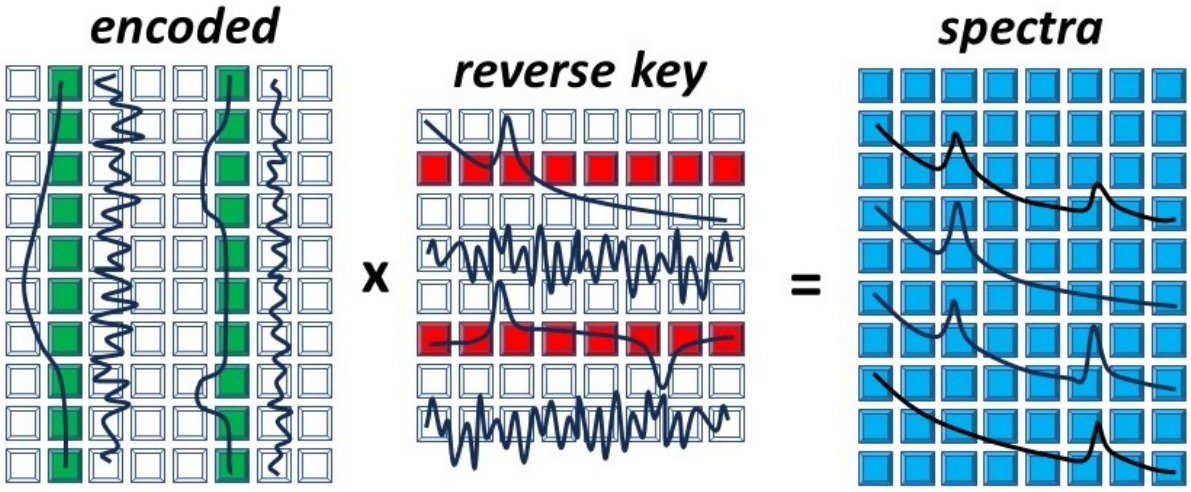
Driven by his intuition, Mr. Bond decides to skip these seemingly useless coding columns. This choice speeds up his transmission work at night and reduces the risk he faces. He informs the MI-6 headquarters that they should retain only a few specified rows in the reverse key matrix when decoding. To his surprise, the headquarters manages to successfully decode the spectra using this reduced set of rows. In fact, they even admit that the quality of the decoded spectra has improved significantly. It appears that much of the data Mr. Bond had been transmitting before was nothing more than noise.

Figure 5: Sparse economical encoding.



On that fateful day, James Bond made a life-altering decision. He stopped stealing, peeling, eavesdropping, and embarked on a completely different path. No longer would he receive key matrices from headquarters; instead, he took matters into his own hands and constructed the keys himself for each dataset.

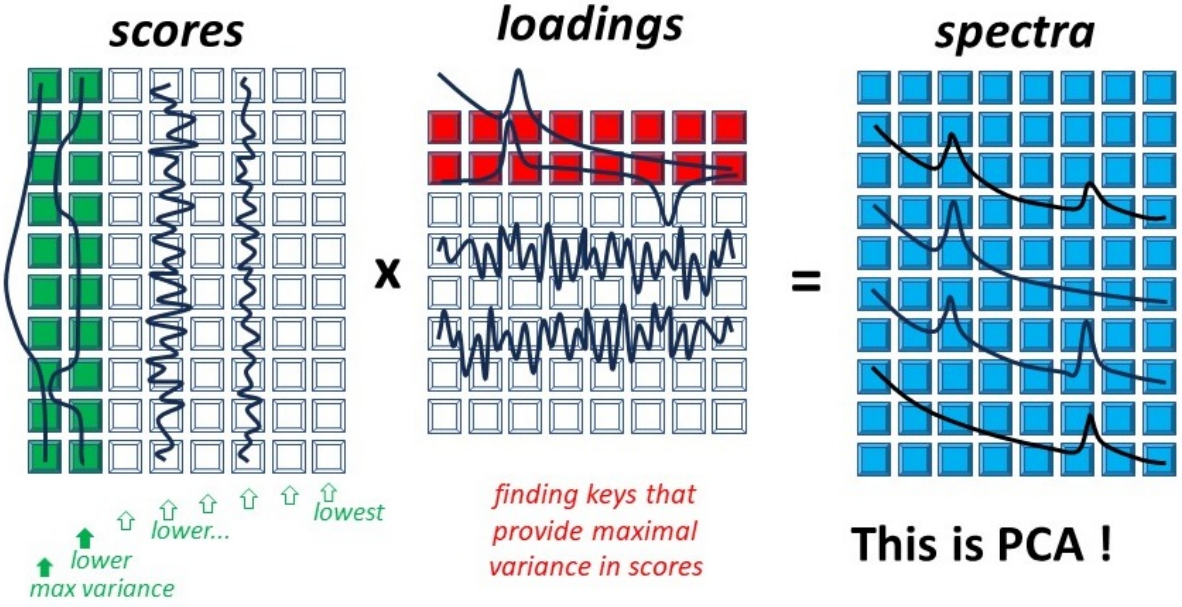
Figure 6: Inverse transformation.



His new mission was clear: to discover basis vectors that would enable him to express the data using the fewest possible coordinates, ultimately compressing the data and improving its quality by reducing noise.

As he delved deeper into this new endeavor, Bond came across a remarkable rule. The key columns that yielded the highest data variance in the columns of the encoded matrix were most efficient for encoding. This criterion of maximizing data variance wasn't the only possible criterion (we can explore other criteria in future discussions), but it proved astonishingly successful. Bond meticulously constructed the rows in the reverse key matrix, sorting them based on their efficiency in producing maximal data variance. He then employed only a few of the top-ranked rows to reconstruct the complete datasets.

Figure 7: Compressed encoding.



With this transition, James Bond inadvertently invented Principal Component Analysis (PCA). He now refers to the red reverse key matrix as the "loadings" matrix, and to the green encoded matrix as the "scores" matrix.

James Bond's role has changed dramatically since that time. He is still employed by the secret service, but now as a data scientist. We are not going to name his employer. To conceal the real name, let's call it by some senseless abbreviation, for instance, 'MI-6'. And James Bond's career in MI-6 develops quite

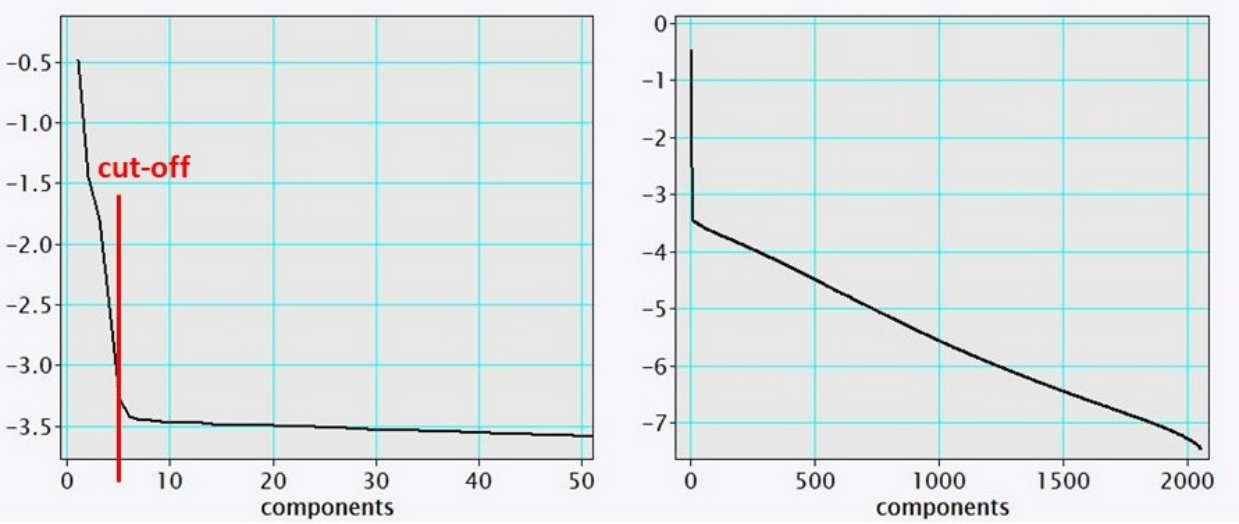
successfully.

1.2 Now closer to the technical topic

For those familiar with PCA, they can skip the essay above and delve right into this paragraph. Is the Bond's invention indeed as magical? How much noise can we remove with PCA? All the noise or just a fraction? The answer is that PCA cannot completely eliminate all noise from a dataset.

Consider the plot of variances in the scores columns, which is called scree plot (although some journal technical editors always tend to correct it for screen plot...). Each column in the score matrix and the companion row in the loadings matrix form a principal component. The scree plot for a typical EELS dataset is shown below. Note that the variances are displayed in the logarithmic scale, therefore the negative values denote just numbers below 1. When constructing the scree plot, it is common practice to calculate and plot the variances for a limited number of principal components, typically the first 20 to 50 as I showed in the left figure. However, for better understanding the things, I also calculated the variance for all 2048 principal component (shown in the right). Yes, the total number of the principal components equals the number of the energy channels, i.e. 2048.

Figure 8: Variances of principal components (screeplot). Left picture shows first 50 components, right one - all 2048 components.



The scree plot helps researchers strike a balance between retaining enough principal components to capture meaningful data variations and reducing the noise contained mostly in less significant components. By selecting a cut-off point, such as the 5th component, we declare: all components at the left (i.e. 1-5th components) are useful while all components at the right (6-2048th) are "noise components" and should be removed. Does it mean that we get rid of all the noise by removing components 6-2048? No way!

I believe Edmund Malinowski (E.R. Malinowski, Anal.Chem. 49 (1977) 606) was the first who clearly showed that the so-called 'meaningful components' also consist of noise. With using a simple assumption of equal distribution of noise in the green 'score' matrix above he calculated how much noise is removed. There is always an 'imbedded' noise in the major principal components, although typically not much. For the shown example, I estimated that 99.5 percent of the total noise is incorporated in components 6-2048 while only 0.5 percent is imbedded in components 1-5. that is not surprising as we compressed the data 2048/5 400 times!

Let's explore the question: What happens if we use more than 5 components for the PCA reconstruction? Would the results significantly worsen? To answer this, let's delve into the calculations. The noise variance is additive among the components, thus we can safely sum it up in any required range. Do not forget to take a square root of this sum in order to rescale it from quadratic deviations to the linear scale! The results are in figure above. As we increase the number of included principal components, we observe a gradual decrease in the amount of noise removed. Initially, this reduction follows an almost linear pattern, but it becomes slower as we include more components. If we utilize 50 components instead of 5 for reconstruction, the amount of removed noise decreases from 99.5 to 95 percents. The question arises: is this reduction

Figure 9: This graph shows how much noise we remove when reconstruct the dataset with a given number of principal components. Left picture shows first 50 components, right one - all 2048 components. Note that we never remove 100 percent of noise.

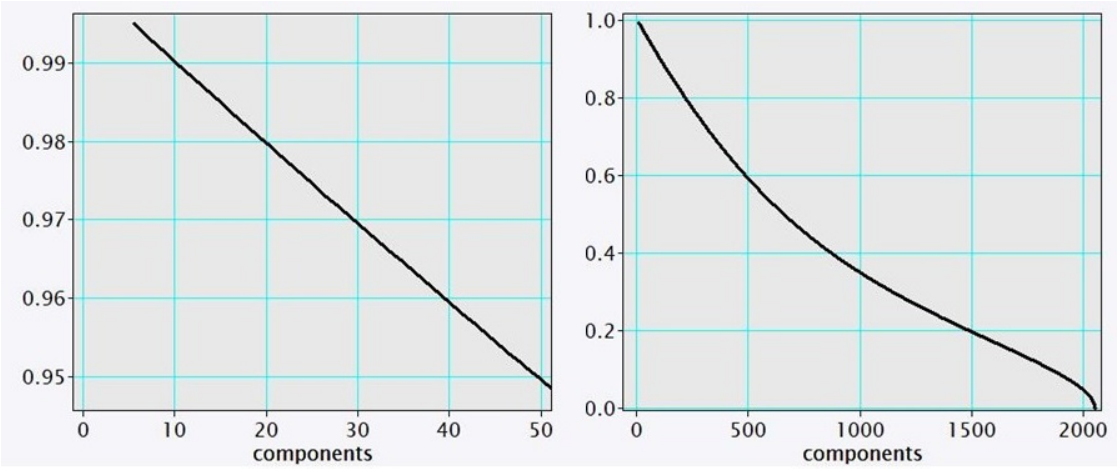
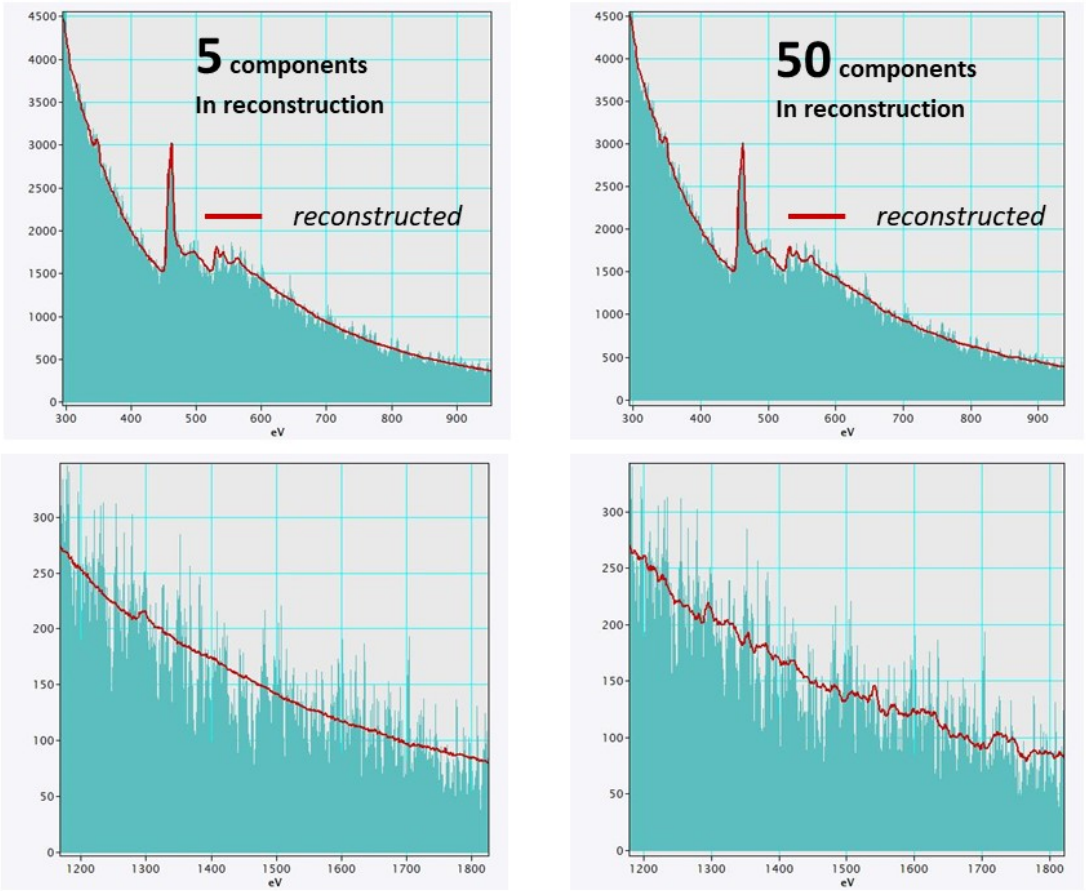


Figure 10: Quality of the denoised curve (red) depending on how many principal components, 5 or 50, was used for reconstruction. The lower energy region is on top and the higher energy region is on bottom.



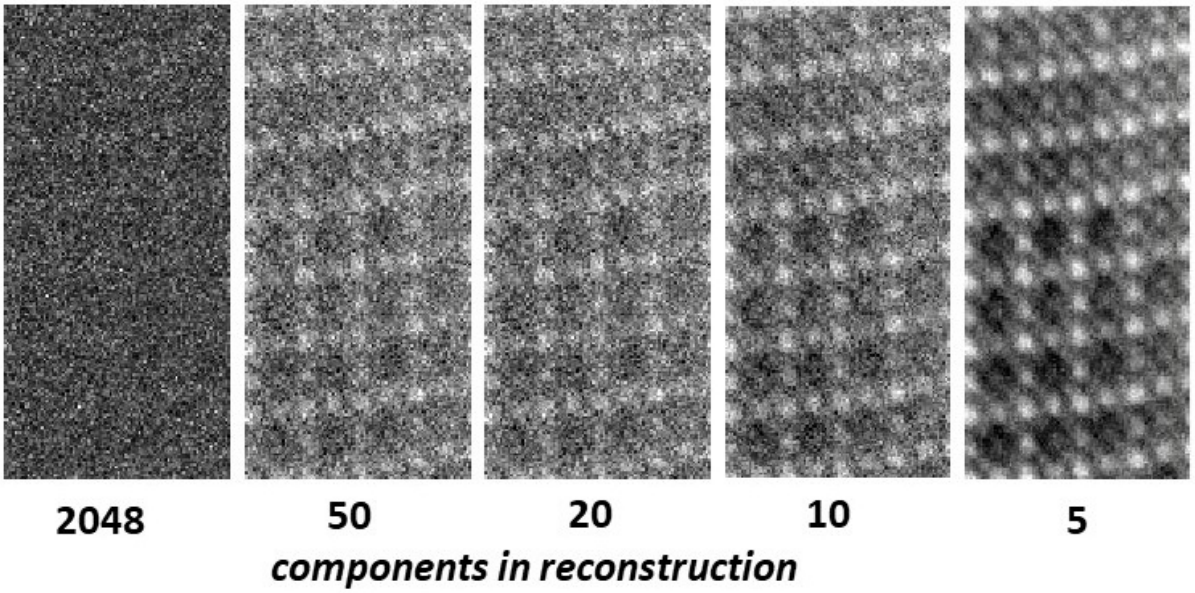
substantial or not? It depends...

The considered EELS spectra exhibit distinct statistics across different energy regions. When examining the Ti L edge region, the reconstructions using both 5 and 50 components yield identical results. However, in the case of the region near the Mg K edge, where the data are heavily affected by noise, the reconstructions

using 5 and 50 components display noticeable differences.

Correspondingly, when we examine the spectrum-image slice at 1880eV with a width of 1eV, we observe significant differences in the reconstruction results obtained using 2048, 50, 20, 10, and 5 components. Note that reconstructing with 2048 components means preserving all possible principal components, which is equivalent to not applying PCA at all.

Figure 11: Energy slice at 1880eV with the width of 1eV as a function of the number of components used in reconstruction.



In summary, keeping an excessive number of principal components during PCA reconstruction gradually diminishes the denoising effectiveness of PCA. However, the question arises: why do people sometimes still use too many components? This topic will be further explored and discussed in upcoming posts.

Lars:
Hello Pavel. Thank you for this contribution, very well written! However, I can't really figure out from your explanation how you came up with 99.5 percent noise reduction. So, you compressed your set by 2048:5 = 410 times. That means the noise should also be compressed 410 times. 100 percent devided by 410 equals 0.24 percent. So, shouldn't it be 99.8 percent noise reduction?! – Am I missing something? Thank you in advance

Pavel:
I afraid its more complicated. First, the noise level itself cannot be additively summed up among components. The noise variance may be summed, then we should take the square root from the summed variance. Second, Malinowski (Anal. Chem 49 (1977) 606) assumed the equal distribution of the noise variance among components, which is not true. In the later article (J. Chemometrics 1 (1987) 33) he introduced some dependence, which still typically underestimated the noise. You can check the article (Chemomentrics Int. Lab. Sys. 94 (2008) 19) to get feeling how complicated might be the distribution of noise among components. I just linearly extrapolated the noise variance from components 10-20 to components 1-5. This way I got 0.5 percent of noise still remaining in the meaningful components, which is still, of course, a very rough estimation.

Juan:
Does it have something to do with the percentage of the explained variance?

Pavel:
No. The explained variance ratio is easy to calculate, however its applicability is limited. Namely, it is useful in the situations of little noise only. Then you can say 'the first 5 principal components explain 99percent of the signal variance, so I may compress the data to 5 components and not loose much. However, the explained variance ratio can be misleading for very noisy data. Imagine a data set with the only noise, no signal variation. Still, PCA will retrieve the noisy components that are a bit more variable than the other noise ones. You can also calculate the explained variance ratio and probably claim that the first 50 components explain 99 percent of variance. But still, these 99 percents are nothing but noise because there is no signal variation in this set. The estimation of the signal : noise proportion in data is a much more complicated task and the starting point here is the theory of Malinowski (Anal. Chem 49 (1977) 606).

2 PCA reveals trends

2.1 James Bond tells the story

This story takes us back to a time when James Bond was sent undercover as an MI-6 agent to a highly classified school. His mission was to observe the participants closely.

Bond meticulously tracked the grades of all the students and created tables where each row represented a student, and each column represented their scores in specific subjects such as math, sports, and geography. Soon, Bond found himself overwhelmed by a massive amount of data. To simplify it, he decided to calculate the average grade for a certain period, thinking it would provide a more representative picture.

However, the results turned out to be inconclusive. Bond then attempted to calculate the average grades across all students. Since this measure was independent of each student's individual abilities, it reflected rather the quality of teaching in the school for each subject. In the next step, Bond realized that the individual deviations from this average would be more informative.

Nevertheless, analyzing the data proved to be challenging. The headquarters advised him to employ some linear algebra techniques, specifically the rotation of basis. You see, the scores in the three subjects can be likened to coordinates in a three-dimensional space, and they can be transformed into a rotated basis.

James experimented with this rotation and made an intriguing discovery. It was possible to rotate the basis in such a way that most of the columns in the transformed table became zero or close to zero. In other words, when the data was projected onto the new y and z coordinates, it resulted in little useful information and mostly represented noise. These y and z columns were deemed unimportant and could be removed.

Figure 12: Composing an average from a number of data matrices.

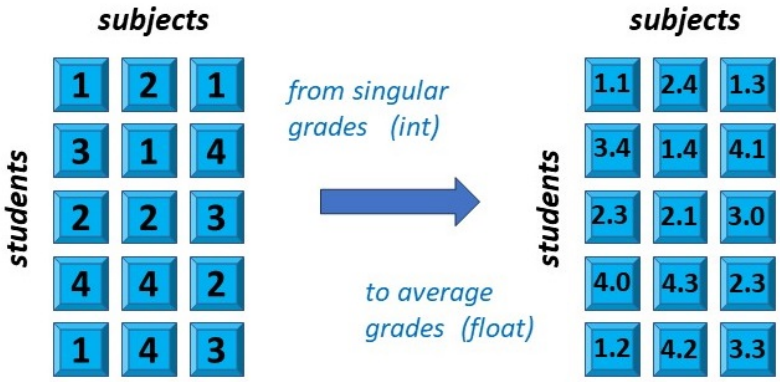


Figure 13: Now the matrix shows the deviation from the average.

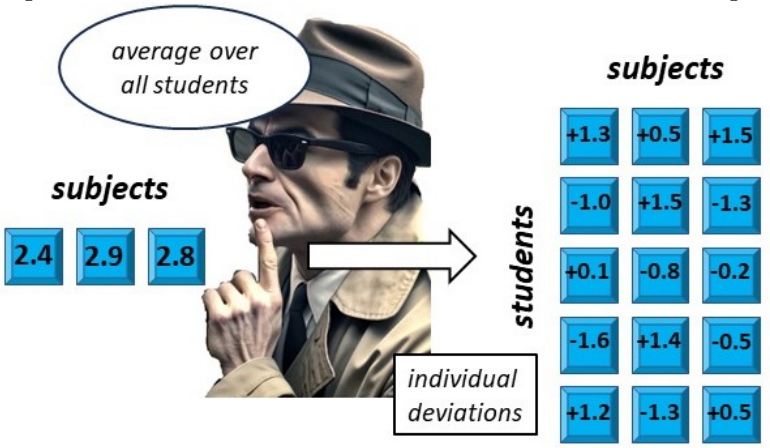
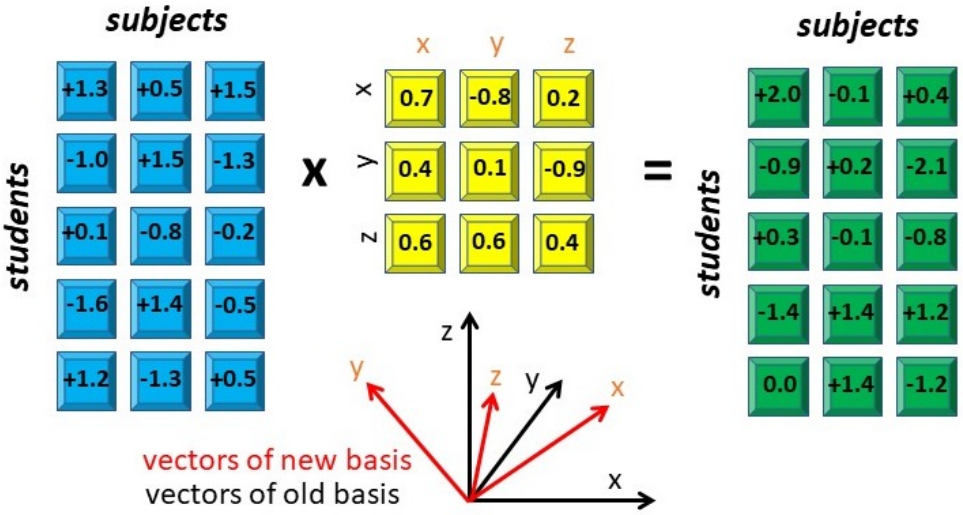
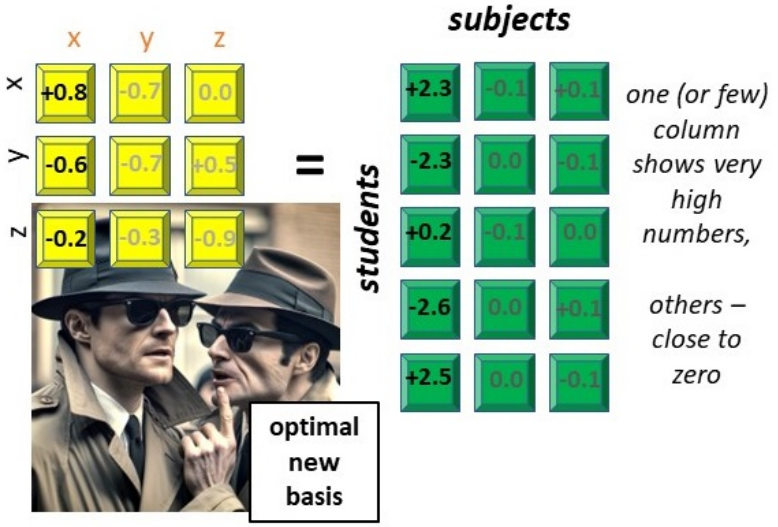


Figure 14: Recast data matrix in a new rotation basis.



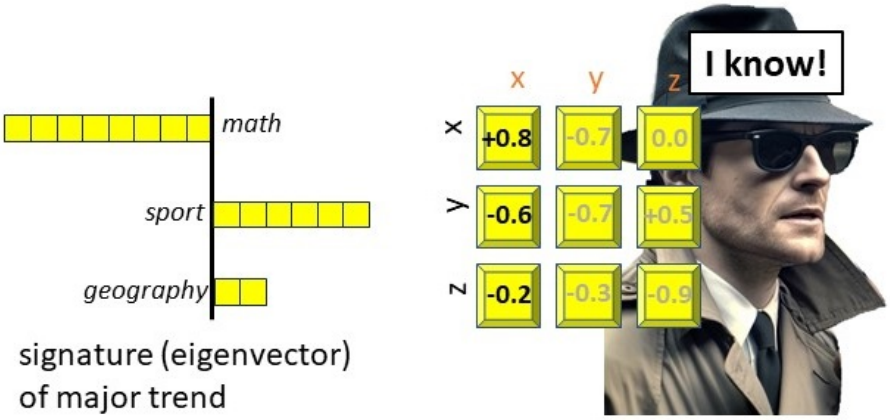
Now, what about the remaining new x coordinate? James Bond found that it exhibited a distinct pattern: positive numbers for math and negative numbers for sports and geography, in a certain proportion. Aha, thought James, students with positive values in this specific score tended to have analytical thinking skills, while those with negative values were more inclined towards activities such as traveling, acting, and shooting.

Figure 15: In the optimal basis, the only few important column may be retained in the matrix while the others may be discarded.



Bond now had a powerful tool to characterize the individual profiles of each student, providing crucial results to report back to headquarters.

Figure 16: Trend has a certain signature: Prominent in math but not in sport and geography are at the one pole of trend while good in sport and geography but weak in math are at the other pole.



2.2 Technical example

How can we apply James Bond's experience to our own endeavors? Let's consider a vast collection of spectra comprising 1000 energy channels, all of which are affected by noise. Behind the scenes, the only variation lies between compound A and compound B, whose ideal spectra are depicted in the figure.

However, the presence of significant noise makes spectra horrible. It is not clear what is going on in the data set:

No panic! Following Bond's strategy, we calculate the mean spectrum and treat all the data as deviations from this mean. Still, analyzing the data with 1000 channels proves challenging. Go further!

Apply Principal Component Analysis (PCA), which is akin to the rotation technique employed by James Bond. Miraculously, PCA reveals that essentially one parameter varies across the data: the proportion of compounds A and B. To emphasize, instead of dealing with 1000 independent counts across 1000 channels, we find that there is only one parameter that predominantly governs all counts. This parameter is the strength of the deviation from the mean while the shape of deviation is characterized by a certain curve revealed by the PCA basis.

Figure 17: Spectra of two compounds.

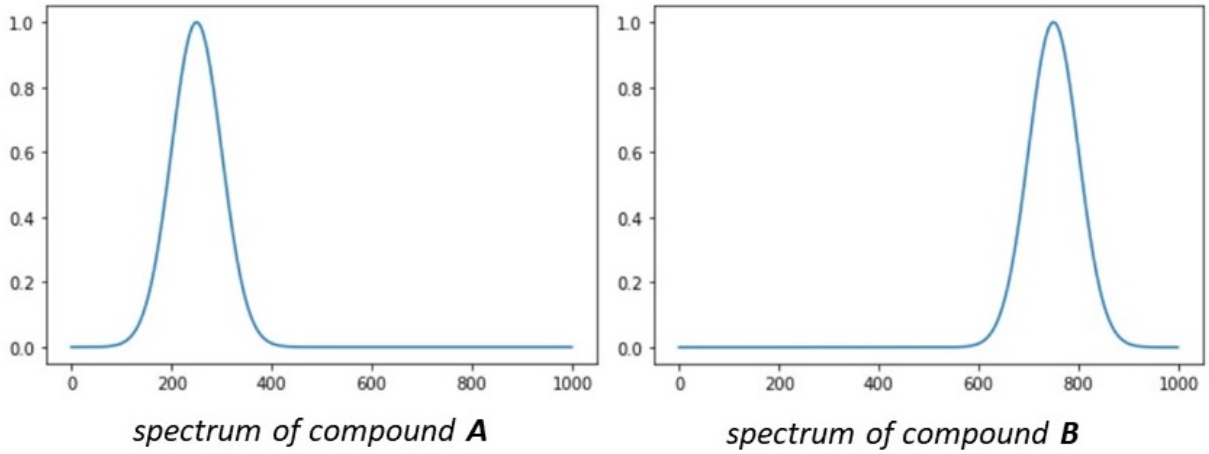


Figure 18: Typical spectra corrupted by noise.

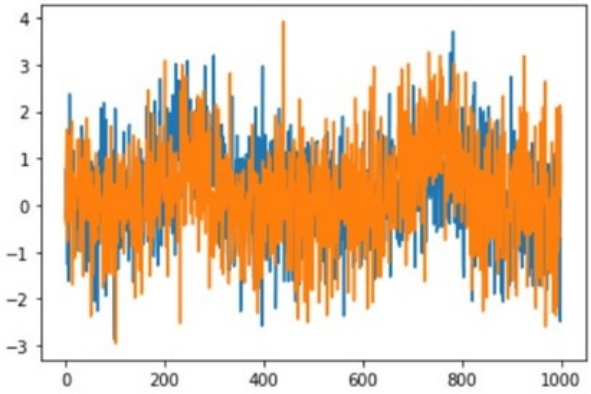
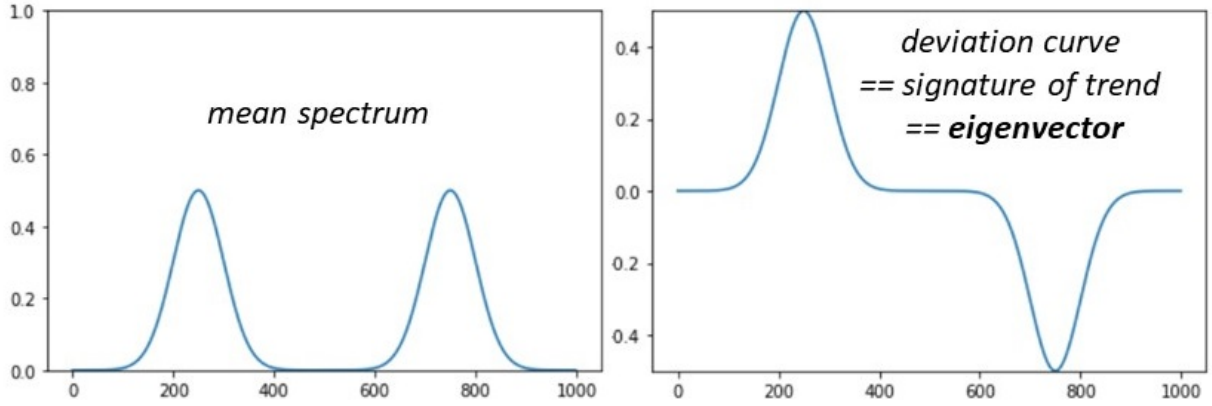


Figure 19: Decompose spectra on mean and deviation.



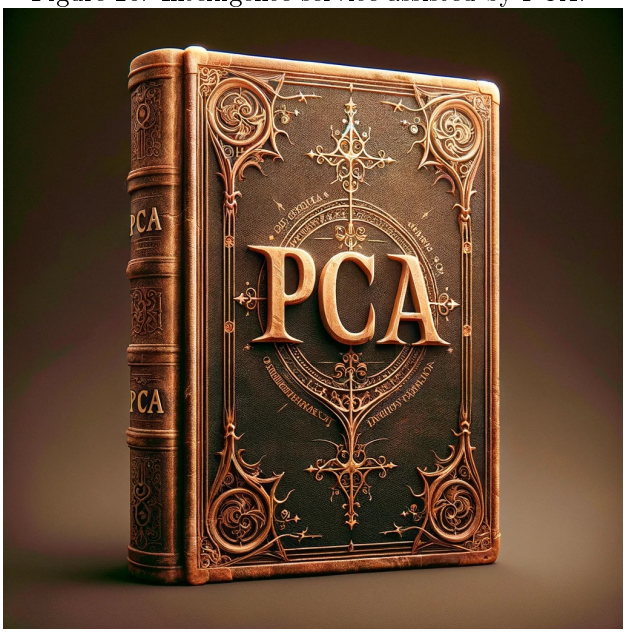
To distance ourselves from spy terminology, let's refer to this curve as an "eigenvector" rather than a "signature." This eigenvector shows a trend.

Lets also clarify our usage of the term "trend," as it deviates from the commonly intuitive definition of collective behavior driven by a specific impulse, such as the buying of Tesla stocks. Instead, we refer to a statistical linear trend, which can be extrapolated in two directions. Following a positive direction signifies the strengthening of a particular feature, while following a negative direction indicates its weakening.

Now, all spectra with a parameter close to +1.0 would exhibit the spectrum of compound A, while those close to -1.0 would display the spectrum of B. Naturally, there are numerous spectra that fall in between A and B, representing a mixture of the two compounds. Their parameter is in the range (-1.0 : +1.0). Our comprehension of the data set has now reached a state of clarity and coherence.

In conclusion, PCA analysis not only enables us to compress and denoise spectroscopic data but also allows us to extract clear variation trends that may go unnoticed to the naked eye. By reducing the complexity of the data and identifying the dominant trends, PCA provides valuable insights and reveals patterns that might otherwise remain hidden.

Figure 20: Intelligence service assisted by PCA.



Abdul:

That's all wrong! People characters can't be described by one parameter.

Pavel:

You are absolutely correct; people are indeed complex beings. I must admit that I oversimplified the story that Bond shared with me. In reality, there were approximately 40 subjects and 4 distinctive parameters involved. From what I recall, these parameters included: 1) "analytical vs acting," 2) "artistic vs technical," 3) "lazy vs hardworking," and 4) "communicative vs reserved." Even with these parameters, one can begin to construct a reasonably accurate profile of an individual. I use the term "in reality," but it remains uncertain what information Bond may have shared. A significant portion of this story still remains top-secret, even to this day. . . .

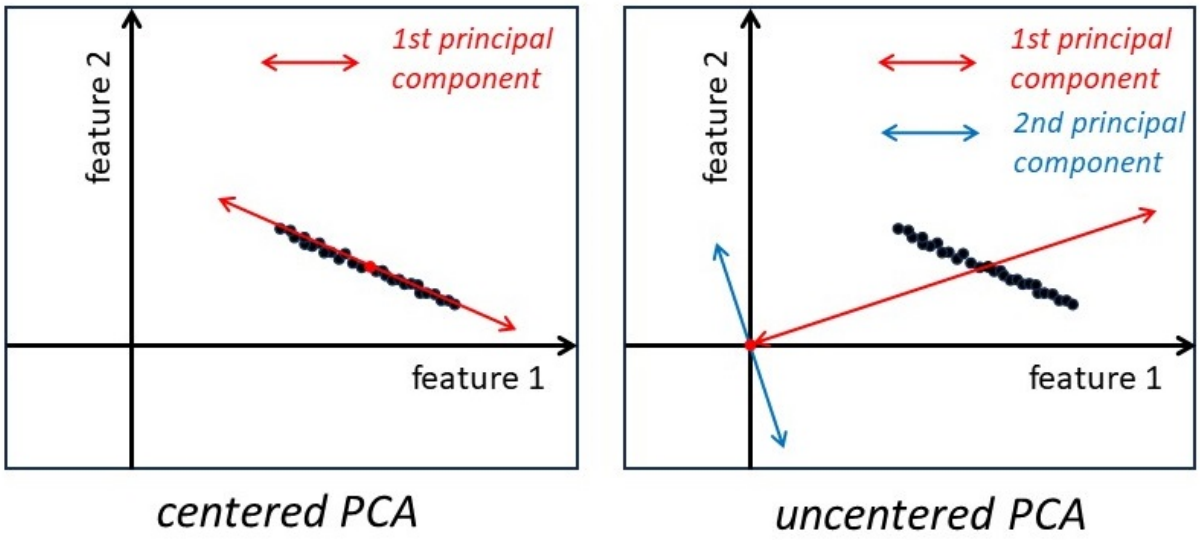
Thomas:

I checked several papers on application of PCA. They generally do not subtract the mean value before the decomposition. If you count everything from the mean, the eigen spectra may be negative like in your last picture. That is hard to understand. Counting from zero, not from mean, is more logical.

Pavel:

Hmm. . . I doubt that most people do not subtract the mean prior to PCA. I believe the most common approach is subtracting the mean, which is called centered PCA. However, you are right, sometimes people ignore centering. This is because they do not attempt to find a trend in the data but simply want to denoise it. I will try to clarify this with a simple example. Suppose there is a dataset with only two energy channels or features. There is a clear trend – the features change in a 2:1 proportion as shown in the figure. Centered PCA will immediately find the direction of this trend, while uncentered PCA will first find the eigenvector pointing more or less to the center of the data distribution. Moreover, the second eigenvector will also not coincide with the true trend because it is restricted to the orthogonality conditions of eigenvectors. Therefore, you end up with two basic vectors, none of which coincides with the true trend. However, it is easy to see that the 'true' trend direction is just a linear combination of these two vectors. Thus, the

Figure 21: All secret agents dressed in the same way are visible to anyone.



denoising reconstruction still works, although you need one more component than in the centered case. I should mention, however, that if there are more than one trend in the data, the situation becomes more complicated and the differences between centered and uncentered PCAs are not significant. To summarize: for the denoising task, centered and uncentered PCAs are almost equivalent, but if you want to retrieve the trend, centered PCA is needed.

2.3 Used codes

Listing 1: Generate model spectra from mixture of two compounds

```
import numpy as np
import matplotlib.pyplot as plt
import math as m

def gaussian(x, mu, sig):
    return np.exp(-np.power((x - mu)/sig, 2.) / 2)
def smooth_gaussian(spec,mu,sigma):
    noise = np.random.normal(mu,sigma,D)
    spec +=noise
    return spec

D =1000 #number of energy channels
SignalW =50 #width of the signal peak in spectrum

spec1 =np.arange(D)
spec1 = gaussian(spec1,D/4,SignalW) #1st compound shows a peak at the 1st quater of spectrum
spec2 =np.arange(D)
spec2 = gaussian(spec2,3*D/4,SignalW) #2nd compound shows a peak at position 3/4 of spectrum

plt.plot(np.arange(D),spec1) #spectral signature of 1st compound
plt.show()
plt.plot(np.arange(D),spec2) #spectral signature of 2nd compound
plt.show()
plt.ylim(0,1)
plt.plot(np.arange(D),(spec1+spec2)/2) #mean spectrum
plt.show()
plt.plot(np.arange(D),(spec1-spec2)/2) #difference spectrum
plt.show()
```

```
Int =2 #spectra intensity constant
Sigma =1 #gaussian noise added to spectra

#generate spectra with different proportion of 1st and 2nd compounds
fract =0.7
spec =(fract*spec1 + (1-fract)*spec2)*Int
spec =smooth_gaussian(spec,0,Sigma)
plt.plot(np.arange(D),spec)

fract =0.3
spec =(fract*spec1 + (1-fract)*spec2)*Int
spec =smooth_gaussian(spec,0,Sigma)
plt.plot(np.arange(D),spec)
plt.show()
```

3 ICA vs PCA

3.1 James Bond tells the story

Once, I asked James Bond what was most important for a secret agent: shooting smartly or running quickly.

"None of them," answered Bond. "The most important thing is to be invisible. You should not be noticed by anybody who is searching for you."

Figure 22: A secret agent should be invisible, otherwise...

"So, should you be dressed as a very average person?"

"Not exactly," replied James. "Indeed, we considered what should be a kind of average clothing style, but we cannot outfit all agents in such a style. A long time ago, our hereditary princess suddenly disappeared during her visit to Rome. A hundred agents were simultaneously dispatched to Rome to find her."

"I think I heard something about that..." I said.

Bond's face suddenly hardened. "You could not have heard about that. It was top-secret."

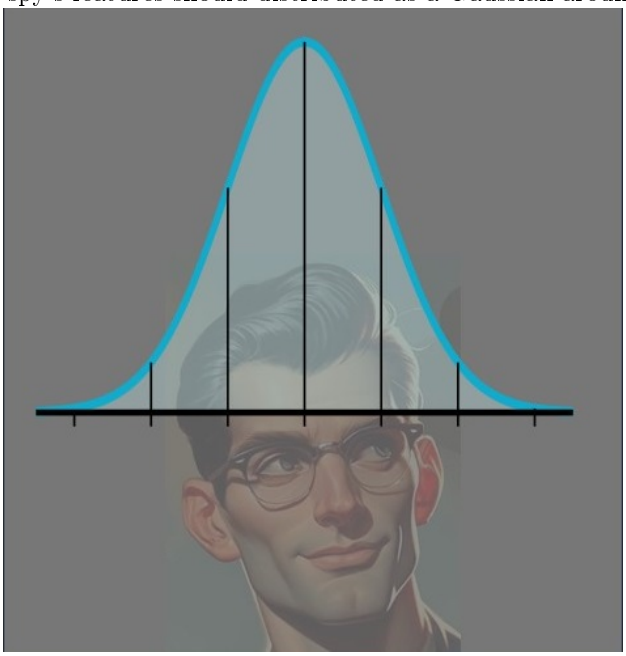
But then he softened a bit. "No matter, it's an old story. So, we sent a hundred agents dressed as 'average,' but they all looked the same. When they disembarked from the plane, all the Italians greeted them with 'buongiorno, signori agenti segreti!' It was a complete fiasco. However, we learned from that case. You should not look 'average'; you should deviate from the norm, but do it in a 'usual' way."

"Some kind of random distribution?" I asked.

Figure 23: All secret agents dressed in the same way are visible to anyone.

"Yes, but not just any random distribution," Bond replied. "You should be distributed like a Gaussian curve around the average. That way, it's difficult to catch you. Have you heard of Independent Component Analysis?"

Figure 24: All spy's features should distributed as a Gaussian around the average.



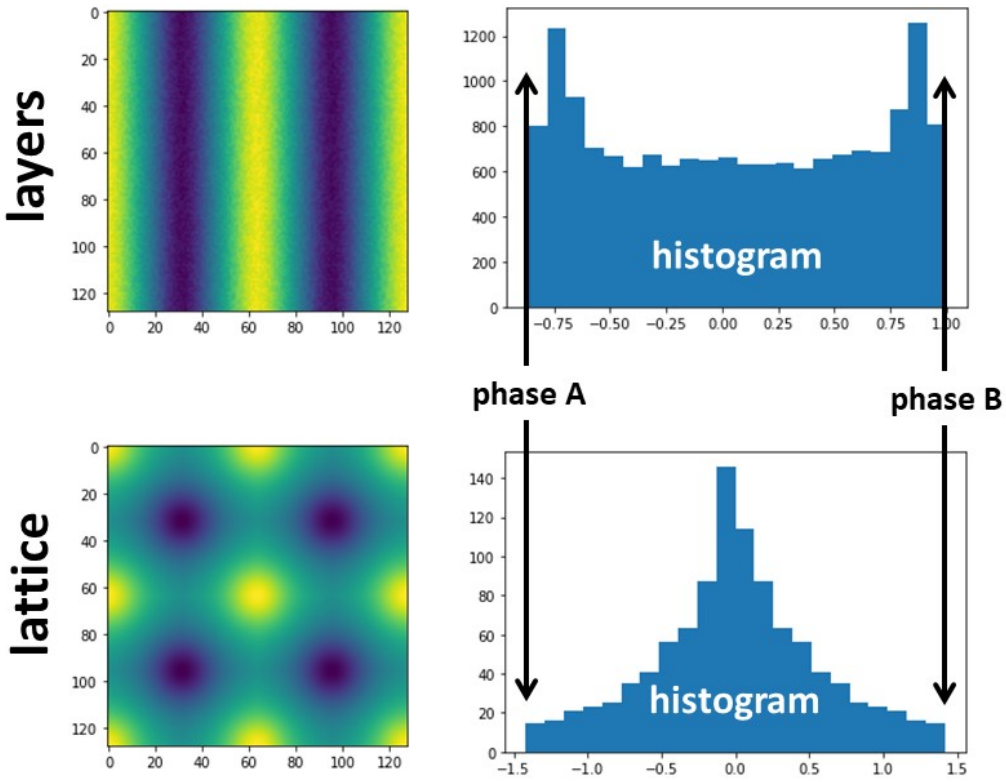
"Is it about terrorists preparing explosions on Independence Day?" I guessed.

"Not exactly," Bond smiled. "It's a technique for identifying features distributed in an unusual way, such as anything that deviates from the Gaussian distribution, which is the most common in this world. Things that don't follow a Gaussian pattern might be of interest to us. For example, detecting speech buried in a sea of noise. In fact, we rely on Independent Component Analysis more often than we do on our guns."

3.2 Apply ICA to materials science

I was fascinated by what James Bond had said about Independent Component Analysis (ICA). I thought, maybe it could be applied to materials science, much like we had previously applied PCA. Perhaps ICA could even outperform PCA? I had come across an article (J.M.P. Nascimento IEEE Trans. Geosc. Remote Sens. 43 (2005) 175) claiming that ICA wasn't very suitable for materials science, but honestly, its arguments didn't convince me.

Figure 25: Two distributions of chemistry: layers and atomic lattice. In both cases, the content varies from A to B but the histograms of distributions are quite different.



So, what is ICA? Like PCA, it involves rotations in a multidimensional factor space, but the way the basis is rotated differs. First and foremost, it's essential to note that ICA always requires PCA as a preprocessing step. It begins by extracting the components with the highest variance, i.e., the principal components. However, it then does something unusual: it normalizes all components to have unity variance. The process called whitening equalizes the components, making it impossible to determine which ones are more principal and which are less. Subsequently, ICA once again rotates the basis, this time aiming to identify components with the highest non-Gaussianity. Mathematically, it can be demonstrated that such components are most likely independent of each other.

You might assume that ICA always produces independent components, while PCA does not. However, that's not entirely accurate. In most cases, PCA components are also independent of each other, whereas ICA doesn't always yield completely independent components. The difference lies more in the algorithms used for rotation within the factor space.

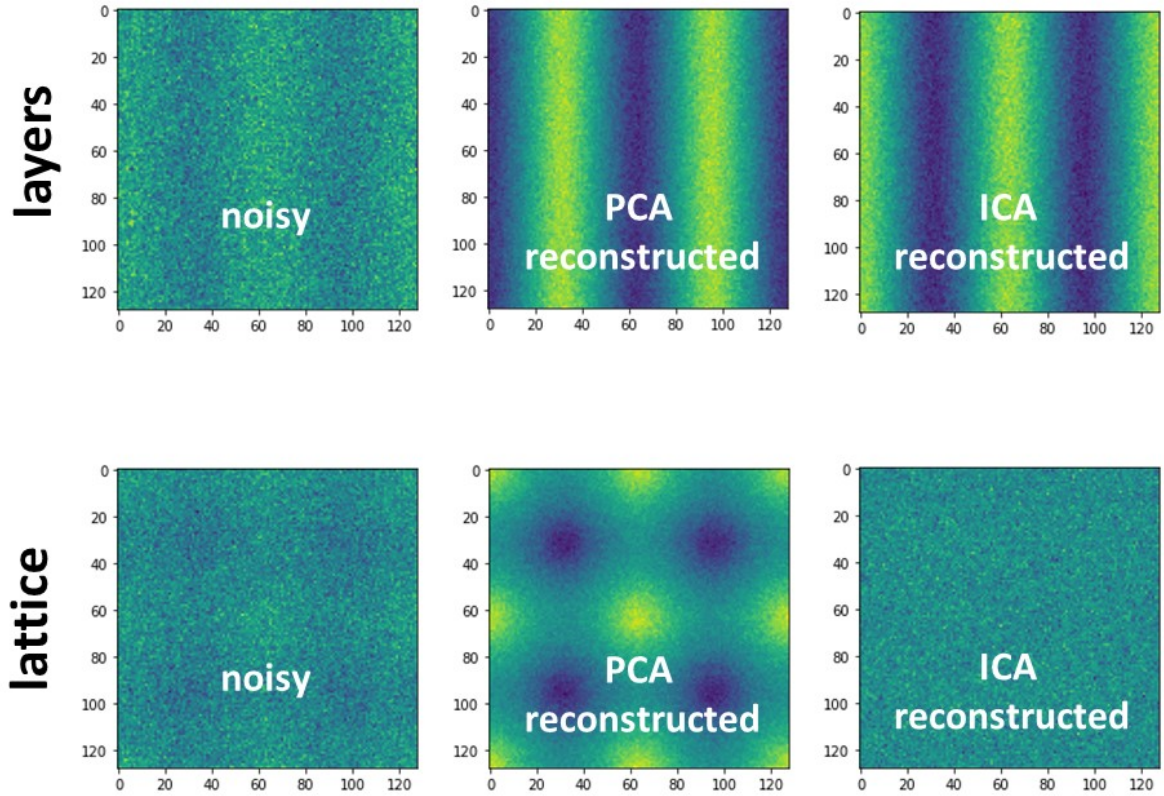
To highlight this difference, I generated two model datasets in which the composition smoothly transitions from phase A to phase B. In the first set, A and B represent layers, while in the second one, they appear as an atomic lattice. The crucial point is that the chemistry distribution from A to B is almost Gaussian for the lattice but strongly non-Gaussian for the layers.

For each pixel of the data, I generated a spectrum (consisting of 1000 channels) corresponding to its

A/B fraction and added a significant amount of noise to obscure the original chemistry. When examining any energy slice of the generated spectrum-images, it becomes challenging to distinguish between A and B due to the noise. To make the maps visible, one must apply either PCA or ICA. Then, I made a remarkable observation: PCA and ICA perform equally well for the layers but not for the lattice. For the lattice, ICA fails entirely. This is because A and B are distributed as Gaussian in the lattice, and ICA cannot differentiate it from noise.

In conclusion, the success of ICA in materials science strongly depends on the inherent data distributions, which can vary widely in the field of materials science. In contrast, PCA does not concern itself with the distributions; it simply captures variations that exceed the noise level. Thus, the application of ICA in materials science is much more limited than, for instance, in speech recognition.

Figure 26: PCA reveals distribution maps in both layers and lattice. ICA does the job for layers while fails for lattice.



Nogami:

I see that PCA and ICA results for layers are not identical. ICA fits better to that shown in a previous picture. Is ICA more accurate here? Thank you very much.

Pavel:

Please take into account that PCA and ICA do not know were is your phase A where is B. They just label components randomly. If you swap colors in the figure, the PCA and ICA results will be identical.

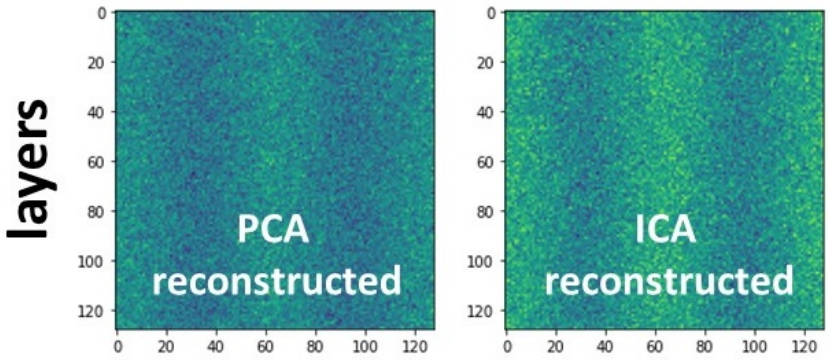
Bernhard:

Thanks for this humorous and enlightening post. It is interesting to see the difference in behavior between the two techniques demonstrated like that. Is there also a counter example where ICA would clearly beat PCA?

Pavel:

That's a good point. I believe it might beat. According my general understanding, a very weak (varying far below the noise level) but strongly non-Gaussian signal could be retrieved by ICA but not by PCA,

Figure 27: Same PCA and ICA treatments but the noise level is increased 8 times.



which is sensitive to the signal variance only. I tried to increase the noise 8 times and repeat PCA and ICA treatment. This time PCA works quite unsurely and ICA can slightly improve its results when rotating 3 principal compounds. However, the effect is unstable as ICA requires PCA as pre-treatment in any case.

3.3 Used codes

Listing 2: Generation of model datasets

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import kurtosis

Size=32

def lor(R,G,C):
    return G/((R/Size*C)**2+(G/2)**2)

def rad(X,Y):
    return np.sqrt(X**2+Y**2)

def gaussian(x, mu, sig):
    return np.exp(-np.power((x - mu)/sig, 2.) / 2)

def signal(D,Start,End,Fract,NoiseSigma):
    Sigmas =5 #+-sigma in range
    Sig =(End -Start)/2/Sigmas
    Mu = (Start +End)/2

    spec =np.arange(D)
    spec = gaussian(spec,Mu,Sig)*Fract

    if NoiseSigma >0:
        spec += np.random.normal(0,NoiseSigma,D)

    return spec

def make_SI_2feature(im,Depth,Sigma):
    Height,Width =im.shape
    imSI =np.zeros((Height,Width,Depth))

    for y in range(Height):
        for x in range(Width):
            #print(x,y)

            Frac1 = (1+im[y,x])/2
```

```
            Frac2 =1-Frac1
            spec = signal(Depth,0,Depth/2,Frac1,Sigma) #add 1st feature
            spec += signal(Depth,Depth/2,Depth,Frac2,Sigma) #add 2nd
            imSI[y,x,:] =spec
    return imSI

#simulate atomic lattice
X,Y=np.ix_(np.arange(Size),np.arange(Size))
G =2.55
C =2.85
cell = lor(rad(X,Y),G,C)
cell -= lor(rad(Size-X-1,Size-Y-1),G,C)

cell2 = np.fliplr(cell)
cell3 = np.flip(cell)
cell4 = np.flip(cell2)
motiv = np.zeros((2*Size,2*Size))
motiv[:Size,:Size] = cell
motiv[:Size,Size:2*Size] = cell2
motiv[Size:2*Size,Size:2*Size] = cell3
motiv[Size:2*Size,:Size:] =cell4

atoms = np.zeros((4*Size,4*Size))
atoms[:2*Size,:2*Size] =motiv
atoms[2*Size:4*Size:,:2*Size] =motiv
atoms[:2*Size,2*Size:4*Size] =motiv
atoms[2*Size:4*Size,2*Size:4*Size] =motiv
atoms /=np.max(atoms)
plt.imshow(atoms)
plt.show()

print('phase A',atoms[0,0],'phase B',atoms[Size-1,Size-1])
dist=cell.flatten()
plt.hist(dist,bins='auto')
print('kurtosis',kurtosis(dist))
plt.show()

#simulate layers
layer =np.ones((4*Size,Size))
rand =0.2*np.random.rand(4*Size,Size) #small deviations to make histogram more realistic
layer1 =layer*(lor(Y,G,C) - lor(Size-Y-1,G,C)) +rand
layer2 =layer*(-lor(Y,G,C) + lor(Size-Y-1,G,C)) +rand
layers =np.zeros((4*Size,4*Size))
layers[:,:Size] =layer1
layers[:,Size:2*Size] =layer2
layers[:,2*Size:3*Size] =layer1
layers[:,3*Size:4*Size] =layer2
layers /=np.max(layers)

plt.imshow(layers)
plt.show()

print('phase A',layers[0,0],'phase B',layers[0,Size-1])
dist=layers.flatten()
plt.hist(dist,bins='auto')
print('kurtosis',kurtosis(dist))
plt.show()

Depth =1000
Sigma =0.5

#spectrum-images from lattice and layers
```

```
layersSI = make_SI_2feature(layers,Depth,Sigma)
plt.plot(np.arange(Depth),layersSI[0,0,:])
plt.plot(np.arange(Depth),layersSI[0,31,:])
plt.plot(np.arange(Depth),layersSI[0,50,:])
plt.show()
plt.imshow(layersSI[:,:,250])
plt.show()
np.save('layers_s0_5',layersSI)

atomsSI = make_SI_2feature(atoms,Depth,Sigma)
plt.plot(np.arange(Depth),atomsSI[0,0,:])
plt.plot(np.arange(Depth),atomsSI[0,31,:])
plt.plot(np.arange(Depth),atomsSI[0,50,:])
plt.show()
plt.imshow(atomsSI[:,:,250])
plt.show()
np.save('atoms_s0_5',atomsSI)
```

Listing 3: PCA and ICA analysis of datasets

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.decomposition import FastICA
import matplotlib.pyplot as plt
from scipy.stats import kurtosis
import math as m

def plot_graph(data):
    L=len(data)
    plt.plot(np.arange(L),data)

def extract_var(data):
    L=data.shape[1]
    V =np.zeros(L)
    for i in range(L):
        var =np.var(data[:,i].flatten())
        V[i] =m.log(var)
    return V

def extract_kurtosis(data):
    L=data.shape[1]
    K =np.zeros(L)
    for i in range(L):
        kurt =kurtosis(data[:,i].flatten())
        K[i] =kurt
    return K

def sort_by_kurtosis(data,kurt):
    L =data.shape[1]
    data_sorted =np.zeros(data.shape)
    for i in range(L):
        MaxKur = np.argmax(abs(kurt))
        data_sorted[:,i] =data[:,MaxKur]
        kurt[MaxKur] =0
    return data_sorted


filename ="atoms_s0_5"#'atoms_s0_5'
X =np.load(filename+'.npy')
Width =X.shape[0]
Depth =X.shape[2]
X.shape =(Width*Width,Depth)
print('input data',X.shape)
```

```
Comp =100 #should be < Depth but large enough
       #to leave enough dimension for ICA to rotate freely

pca = PCA(n_components=Comp)
pca.fit(X)
Y =pca.transform(X)
print('PCA data',Y.shape)
Var =extract_var(Y)
plot_graph(Var)
plt.show()

ica = FastICA(n_components=Comp,max_iter=1000)
ica.fit(X) #LONG!!!,
Z =ica.transform(X)
print('ICA rotated data',Z.shape)
Kurt = extract_kurtosis(Z)
plot_graph(Kurt)
ZS =sort_by_kurtosis(Z,Kurt)
KurtS = extract_kurtosis(ZS)
plot_graph(KurtS)
plt.show()

Y.shape =(Width,Width,Comp)
PCAimage = Y[:,:,0]
plt.imshow(PCAimage)
plt.show()
np.save(filename+'_pca.npy',Y)


ZS.shape =(Width,Width,Comp)
ICAimage = ZS[:,:,0]
plt.imshow(ICAimage)
plt.show()
np.save(filename+'_ica.npy',Y)
```

4 Accuracy of PCA

4.1 James Bond tells the story

Once, I asked James Bond where his most difficult mission took place—was it in Turkey, Mexico, or Russia?

"In Great Britain, when I was promoted to the central analytical office of MI-6," he answered.

"Were the headquarters suddenly attacked by an army of foreign spies?"

"I would have wished for that. Instead, I had to read endless reports from other secret agents abroad and try to understand what was going on."

"Was that so difficult?"

"Mission impossible. Imagine this: some agent informs us that State X is planning to attack the UK on a certain date with all its ground, naval, and air forces. Such an important message requires verification. Another agent confirms the dreadful plans of State X but claims they intend to attack State Y, not the UK. The third agent shares with us that State Y will indeed be soon invaded, but by State Z, not State X. What would you do when receiving hundreds of such messages?"

"I would kill myself. "

"I was on the verge of doing that. However, my older colleague advised me to relax, as he thought nobody was going to attack. We then developed a specific approach to understand the agents' reports. You know, each agent is 100% confident in their information. However, this confidence is often subjective. We refer to this subjectivity as noise. Moreover, competing countries can intentionally fabricate and issue disinformation, which we call bias. All we need to do is collect a vast amount of data, calculate the noise (subjectivity) distribution, subtract the bias (disinformation), and extract the truth."

"You're suggesting that you calculate this using specific formulas? What might they look like?"

Figure 28: James Bond carefully evaluates all available information.

"No further comments. I can only provide you with one reference (Secret reference). If you read it carefully, you might gain an impression of how we handle big data."

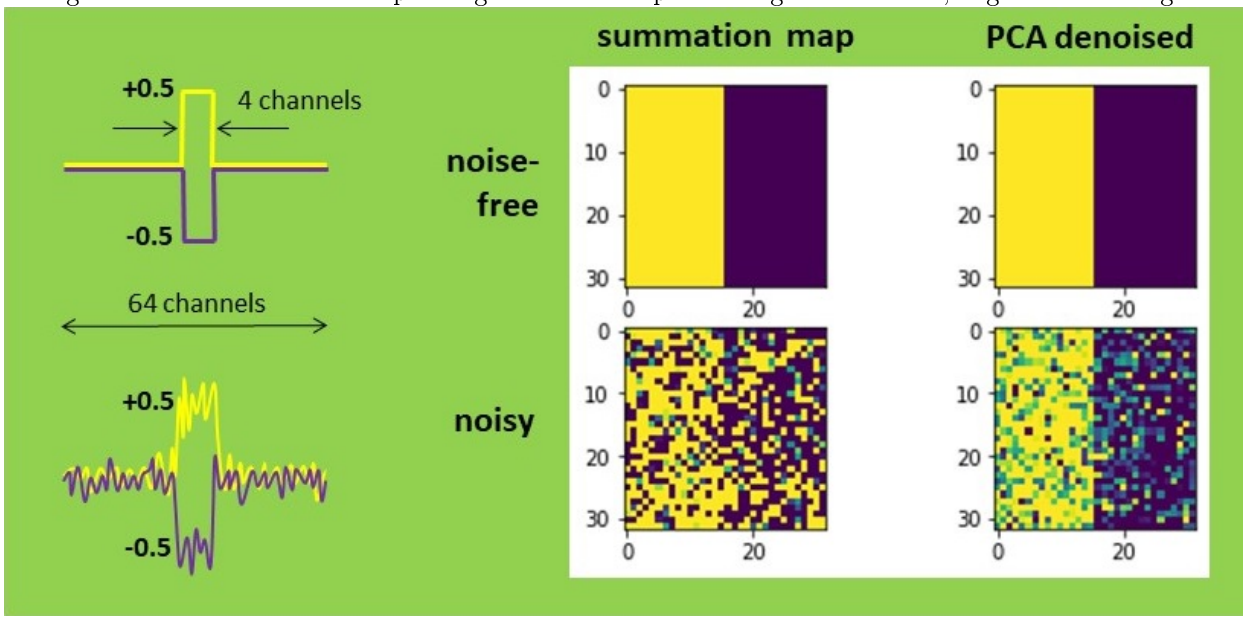
4.2 Evaluate accuracy of PCA

This conversation completely changed my understanding of how a secret service functions in reality. To delve deeper into the topic, I pursued Bond's reference discovering an article by Noaz Nadler, a mathematician at the Weizmann Institute of Science. I thoroughly studied the article and attempted to verify its findings with some simulated spectrum-images.

First, I constructed a simple dataset consisting of 32 by 32 pixels, each comprising 64 spectroscopic channels. The signal exhibited a rectangular shape and could be either positive or negative, resembling a characteristic line of a certain chemical element being emitted or absorbed. This signal variation is symmetric with a zero mean, simplifying the estimation process. The left half of the set was intended to represent the positive signal, while the right half represented the negative signal. One can map such a signal by summing all signal channels or by attempting to retrieve its distribution with PCA. Both methods yield identical results in the absence of noise. However, when Gaussian noise is introduced, PCA significantly

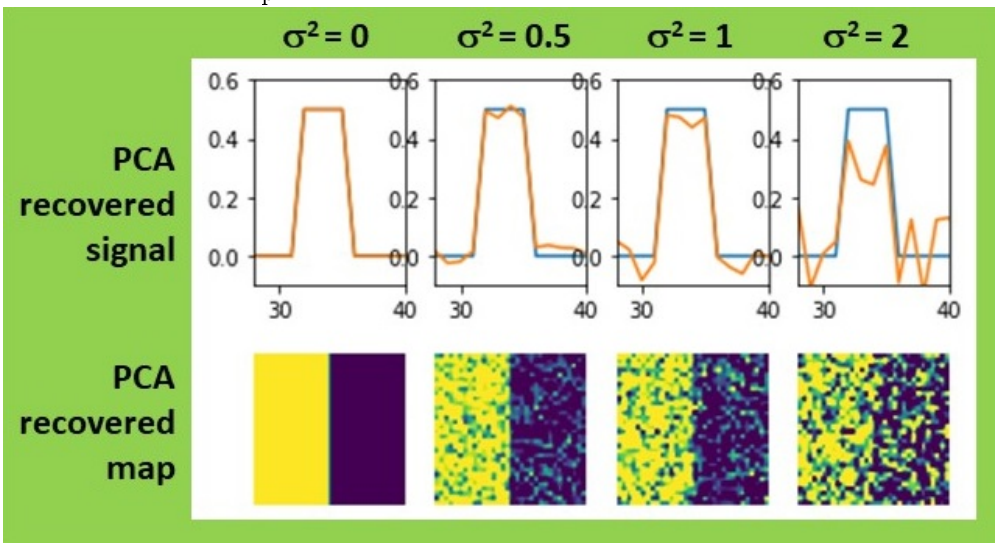
outperforms simple summation. The result is still not perfect, but it appears quite reasonable.

Figure 29: Dataset with a simplest signal variation: positive signal at the left, negative at the right.



Now, a question arises: what is our criterion for estimating the accuracy of PCA? The simplest approach is to compare the shape of the PCA-recovered signal with the true one, which is precisely known in this case. Let's sum the quadratic deviations over all channels and denote it as Δ^2. Note that the appearance of the PCA-reconstructed maps correlates nicely with such a criterion: fewer deviations in the signal shape result in less noisy maps.

Figure 30: Signal profile is restored not perfectly by PCA. More its shape deviates from the true reference, more noisy are reconstructed maps.

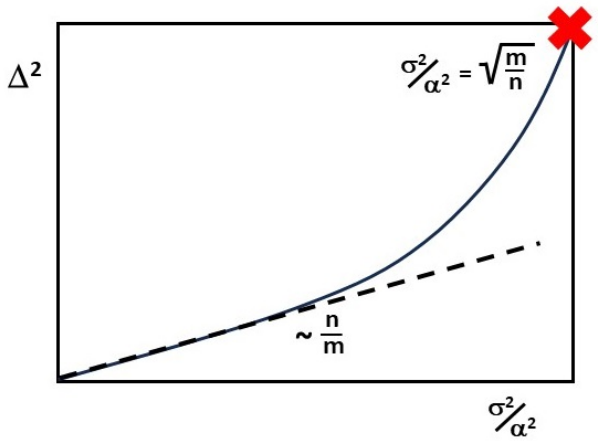


The cited paper suggests that PCA accuracy depends on the following parameters: the number of pixels m (32x32=1024 in our case), the number of channels n (64), the variance of noise σ^2, and the variance of the true, noise-free signal α^2. Calculating α^2 might not be straightforward. I'll just mention that it is exactly **1** in our case. Those interested in verifying this should refer to *Exercise 1*.

The paper then demonstrates that an error in PCA reconstruction is described very simply:

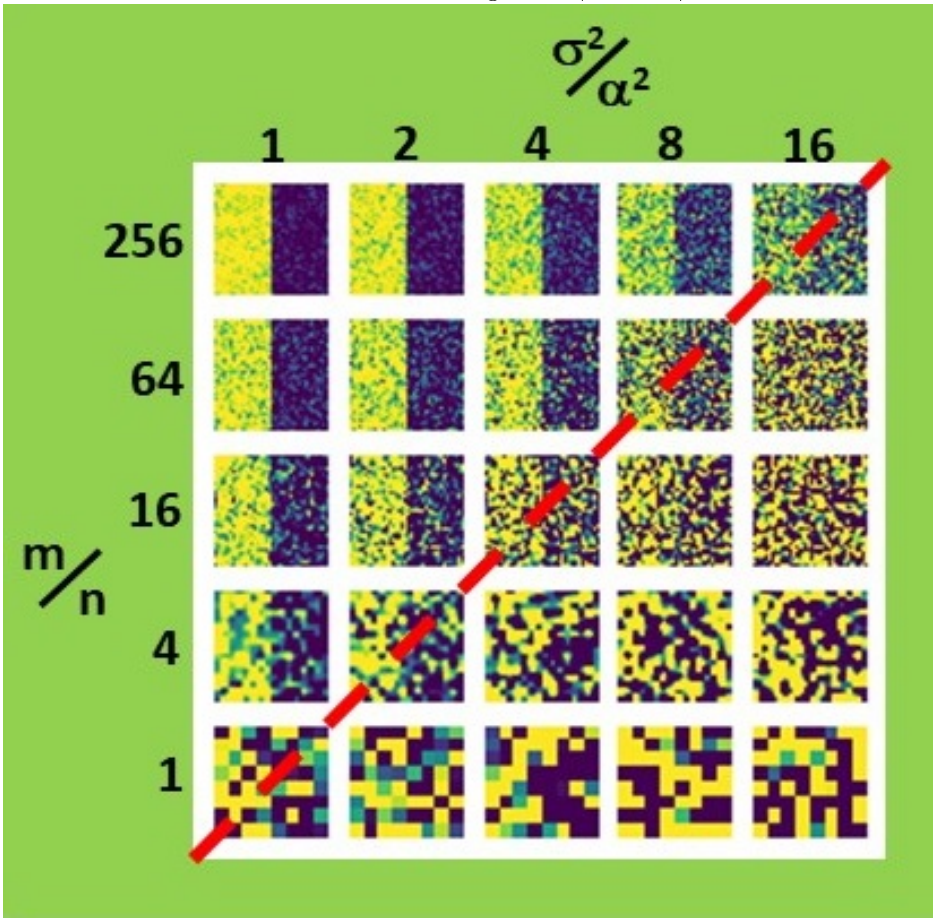
$$\Delta^2 = \frac{n}{m}\frac{\sigma^2}{\alpha^2} \tag{1}$$

Figure 31: Deviation of the PCA-reconstructed signal shape from the truth as a function of noise σ^2.



However, this simplicity holds true only for small σ^2. The subsequent statement of the cited paper is even more instructive. When σ^2 reaches a certain threshold, accuracy collapses to zero, Δ^2 is undefined. There is no useful information in the dataset anymore; at least, nothing can be extracted by such a powerful method as PCA. This situation is reminiscent of James Bond's troubles when too many contradictory reports actually provide no useful information.

Figure 32: PCA-reconstructed maps when varying noise (columns) and number of pixels(rows).



The threshold for the loss of information is:

$$\frac{\sigma^2}{\alpha^2} = \sqrt{\frac{m}{n}} \tag{2}$$

At this point, I apologize for inundating you with too many formulas. However, as you see in the modern world, even secret agents are increasingly relying on formulas rather than old-fashioned master keys in their work.

Now, back to our business! I validated the theory by altering the number of pixels m and the noise level σ^2 in my dataset. In all cases, PCA accuracy improved as m increased or σ^2 decreased, precisely as predicted. Moreover, when the combination of parameters reached the magic Nadler ratio (2), the maps became irretrievable.

For those interested in further testing the Nadler model, I have prepared *Exercises 2 and 3*. The Python codes are attached. Have fun!

4.3 Used codes

Listing 4: Module of standard functions used in this section

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import math as m

def add_gaussian_noise(SI, Mu, Sigma2):
    Sigma =m.sqrt(Sigma2)
    return (SI + np.random.normal(Mu,Sigma,SI.shape))

#consruct SI dataset with zero mean
def build_dataset(SI,Signal_ch,Signal):
    Pixels = SI.shape[0]
    Channels = SI.shape[2]
    Channel_I =int(Channels/2)
    Channel_F =Channel_I + Signal_ch

    SI[:,:int(Pixels/2),Channel_I:Channel_F] =Signal
    SI[:,int(Pixels/2):Pixels,Channel_I:Channel_F] =-Signal

    return SI

#makes PCA decomposition with 'Comp' components
#and return maps of these components
def make_pca(SI,Comp,return_vectors=False):
    Height,Width,Depth =SI.shape
    SI.shape =(Height*Width,Depth)
    pca = PCA(n_components=Comp)
    pca.fit(SI)
    maps =pca.transform(SI)
    maps.shape = (Height,Width,Comp)
    if return_vectors ==True:
        evec =pca.components_
        return maps,evec
    else: return maps

#supress amigioity of the component sign
def swop_comp(denoised):
    Width =denoised.shape[1]
    left_half =denoised[:,:int(Width/2)].copy()
    if np.mean(left_half) <0:
        denoised[:,:int(Width/2)] = denoised[:,int(Width/2):]
        denoised[:,int(Width/2):] =left_half
    return denoised
```

```
#supress amigioity of the eigenvector sign
def swop_evec(evec,Signal_ch):
    Channels =len(evec)
    Middle = int(Channels/2)
    if np.mean(evec[Middle:Middle+Signal_ch]) <0:
        evec = -evec
    return evec
```

Listing 5: Integrated and PCA-reconstructed maps for the noise-free and noisy sets.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *

Channels =64
Pixels =1024
Signal_ch =4
Signal =0.5

Size= int(m.sqrt(Pixels))
SI = np.zeros((Size,Size,Channels))
SI = build_dataset(SI,Signal_ch,Signal)

fig, axs = plt.subplots(2, 2)
images = []

Middle = int(Channels/2)

#noise-free dataset
map_noise_free = np.sum(SI[:,:,Middle:Middle+Channels],axis=2)
images.append(axs[0,0].imshow(map_noise_free,vmin=-1,vmax=1))
denoised =make_pca(SI,1)[:,:,0]
denoised = swop_comp(denoised)
print('noise-free set: variance of pca component',np.var(denoised))
images.append(axs[0,1].imshow(denoised,vmin=-1,vmax=1))

#add noise dataset
Sigma2=1
SI.shape =(Size,Size,Channels)
SI = add_gaussian_noise(SI,0,Sigma2)
map_noise = np.sum(SI[:,:,Middle:Middle+Channels],axis=2)
images.append(axs[1,0].imshow(map_noise,vmin=-1,vmax=1))
denoised =make_pca(SI,1)[:,:,0]
denoised = swop_comp(denoised)
print('noisy set: variance of pca component',np.var(denoised))
images.append(axs[1,1].imshow(denoised,vmin=-1,vmax=1))

plt.show()
```

Listing 6: Correlation between PCA-reconstructed signal shapes (eigenvectors) and PCA-reconstructed maps.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *

Channels =64
Pixels =1024
Signal_ch =4
Signal =0.5
```

```
Size= int(m.sqrt(Pixels))

fig, axs = plt.subplots(2, 4)
images = []

#loop changing noise
Signal_ch =4
Signal =0.5

Sigma2 =0.5
for i in range(4):
    if i==0: Sigma2=0
    elif i==1: Sigma2=0.5
    else: Sigma2 *=2
    print('Sigma2',Sigma2)

    SI = np.zeros((Size,Size,Channels))
    SI = build_dataset(SI,Signal_ch,Signal)
    SI = add_gaussian_noise(SI,0,Sigma2)
    denoised,evec =make_pca(SI,1,return_vectors=True)
    evec=evec.flatten()
    evec =swop_evec(evec,Signal_ch)
    if i==0: rvec=evec
    denoised = swop_comp(denoised)
    images.append(axs[0,i].plot(np.arange(Channels),rvec))
    images.append(axs[0,i].plot(np.arange(Channels),evec))
    axs[0,i].set_xlim([28,40])
    axs[0,i].set_ylim([-0.1,0.6])
    images.append(axs[1,i].imshow(denoised,vmin=-1,vmax=1))
    axs[1,i].set_axis_off()

plt.show()
```

Listing 7: PCA-reconstructed maps when noise (columns) and number of pixels (rows) are varied.

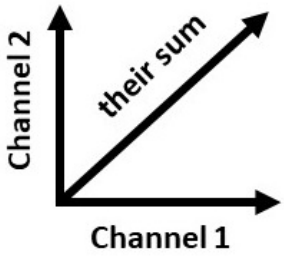
```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *

Channels =64
Signal_ch =4
Signal =0.5

fig, axs = plt.subplots(5, 5)
fig.subplots_adjust(left=0.25,right=0.75)
images = []

#loop changing Sigma2 from 1 to 16 (columns)
#             Pixels from 16384 to 64 (rows)
Pixels =64
for j in range(5):
    if j>0:
        Pixels *=4
    Sigma2 =1
    Size= int(m.sqrt(Pixels))
    for i in range(5):
        if i>0: Sigma2 *=2
        print('pixels',Pixels,'sigma2',Sigma2)

        SI = np.zeros((Size,Size,Channels))
        SI = build_dataset(SI,Signal_ch,Signal)
```

```
        SI = add_gaussian_noise(SI,0,Sigma2)
        denoised =make_pca(SI,1)[:,:,0]
        denoised = swop_comp(denoised)
        images.append(axs[4-j,i].imshow(denoised,vmin=-1,vmax=1))
        axs[4-j,i].set_axis_off()

plt.show()
```

4.4 *Exercise 1*: Variance of noise-free PCA component

You can perceive the counts at each spectroscopic channel as a kind of vectors in multidimensional space. Since the channels are independent, these vectors form the orthogonal basis. PCA identifies the direction of the highest variance of the signal, which, in our case, would be the vector summation of the channel vectors.

Figure 33: Contributions from different spectroscopic channels are summed vectoraly in the principal component.



As wisely noted by Pythagoras, the resulting squared length is merely the sum of the squared lengths of the individual contributions. For each set of four channels, we have a signal of 0.5. Their Pythagorean summation gives $4 * (0.5)^2 = 1$. This represents the variance along the direction of the 1st principal component. You can verify this in the listing of the provided Python script for the noise-free set.

4.5 *Exercise 2*: Effect of the number of channels on the PCA accuracy

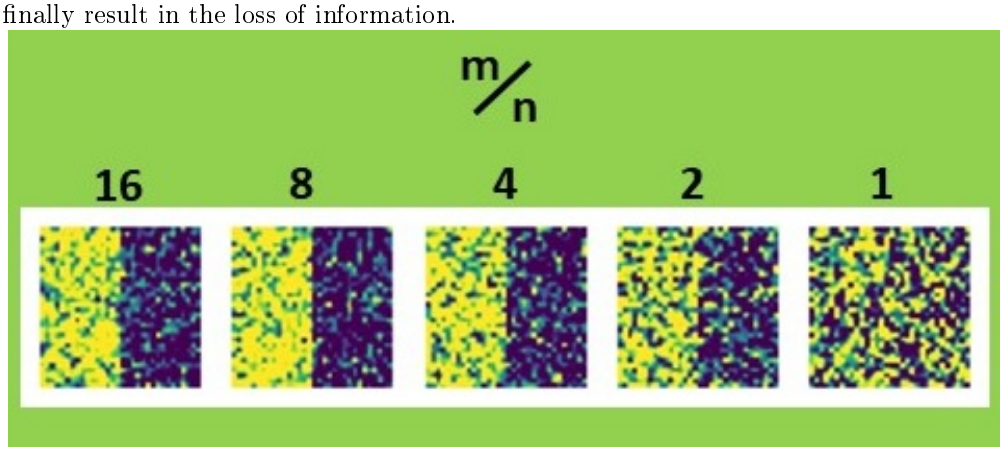
According to formula (1), we might anticipate that PCA error will increase with an increase in the number of spectroscopic channels n. However, this is a fictive dependence resulting from our definition of the cumulative error Δ^2. We defined Δ^2 as the sum of squared deviations over all available channels, whereas, in reality, our interest lies only in the channels where the signal appears. Thus, nothing fundamentally changes as long as Δ^2 is proportional to n/m.

However, this proportionality breaks down with a further increase in σ^2 or n/m. Upon reaching the threshold (2), the signal once again becomes irretrievable.

Listing 8: PCA-reconstracted maps when number of channels is varied.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *

Pixels =1024
Signal_ch =4
Signal =0.5
Sigma2 =1

Size= int(m.sqrt(Pixels))

fig, axs = plt.subplots(1, 5)
images = []
```

Figure 34: Increase of the number of analysed channels n initially has a little effect on the PCA-reconstructed maps but finally result in the loss of information.




```
#loop changing Channels from 64 to 16384 (columns)
Channels =64
for i in range(5):
    if i>0: Channels *=2
    print('Channels',Channels)

    SI = np.zeros((Size,Size,Channels))
    SI = build_dataset(SI,Signal_ch,Signal)
    SI = add_gaussian_noise(SI,0,Sigma2)
    denoised =make_pca(SI,1)[:,:,0]
    denoised = swop_comp(denoised)
    images.append(axs[i].imshow(denoised,vmin=-1,vmax=1))
    axs[i].set_axis_off()

plt.show()
```

4.6 *Exercise 3*: Effect of the spectra dispersion on the PCA accuracy

You know, experimentalists always have the option to alter the dispersion of the spectrometer, such as reducing the covered energy/wavelength range while improving the resolution. The crucial question is, how will this affect accuracy of PCA?

If we double the number of channels for the registered signal, the counts at each channel will be halved, and their squares will be reduced by a factor of 4. However, since we have twice as many channels, the summed variance of the principal component α^2 will only be reduced by a factor of 2.

It gets more complicated with the noise. Assuming the Poissonian nature of the noise and a large number of counts, the variance of the noise σ^2 will increase by a factor of 2. According to formula (1), PCA accuracy degrades by a factor of 4. However, this is not entirely correct if we are interested only in the integral signal. We now have twice as many channels to integrate, so the accuracy of the total signal extraction will be degraded by a factor of 2, not 4.

Nevertheless, the threshold (2) will be met with a noise variance four times smaller than before because the collapse of information is independent on how many channels we intend to integrate afterward.

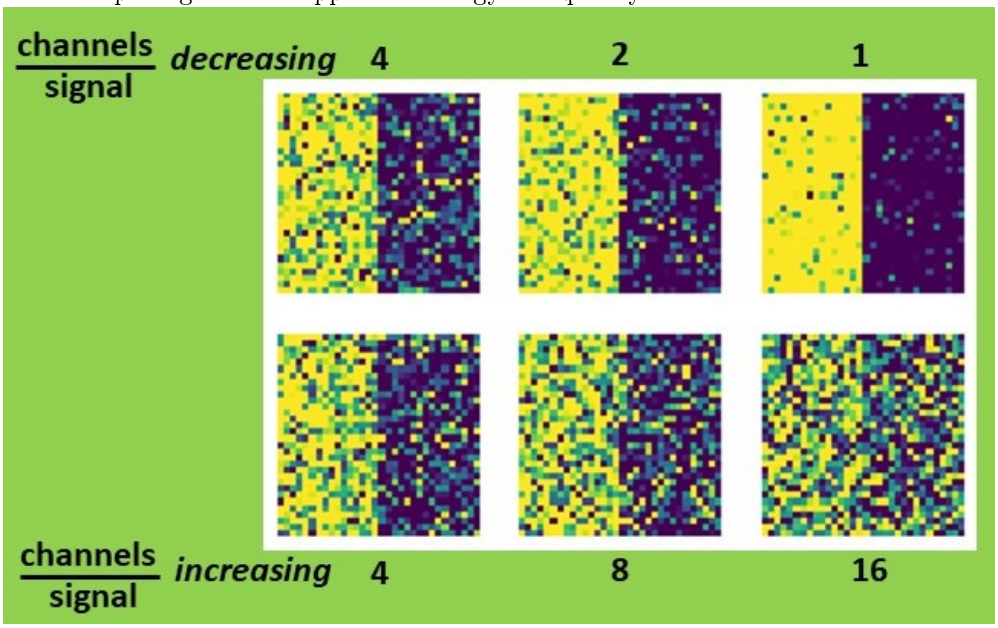
Similar exercises can be conducted by decreasing the number of channels per signal.

The conclusion is following: for a better extraction of the total signal, it is more favorable to reduce the number of channels per signal. This is probably not very surprising, as such a strategy of noise reduction is beneficial even if you do not apply PCA.

Of course, if you are interested in the shape of the signal, not only in its strength, you must keep a certain number of channels available. However, set this number at the minimum if you want to combat noise effectively.

4.7 Used codes

Figure 35: In case you wish to register the total signal strength (not signal shape), it is worse to reduce the number of channels per signal. The opposite strategy will quickly result in the information loss.



Listing 9: PCA-reconstructed maps with changing spectrometer dispersion.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *

Channels =64
Pixels =1024
Sigma2 =1

Size= int(m.sqrt(Pixels))

fig, axs = plt.subplots(2, 3)
images = []

#loop changing dispersion from 4 channel/signal to 1 channel/signal (columns)
Signal_ch =4
Signal =0.5
for i in range(3):
    if i>0:
        Signal_ch =int(Signal_ch/2)
        Signal *=2
    print('Signal_ch',Signal_ch,'Signal','alpha2',Signal_ch*Signal**2)

    SI = np.zeros((Size,Size,Channels))
    SI = build_dataset(SI,Signal_ch,Signal)
    SI = add_gaussian_noise(SI,0,Sigma2)
    denoised =make_pca(SI,1)[:,:,0]
    denoised = swop_comp(denoised)
    images.append(axs[0,i].imshow(denoised,vmin=-1,vmax=1))
    axs[0,i].set_axis_off()

#loop changing dispersion from 4 channel/signal to 16 channel/signal (columns)
Signal_ch =4
Signal =0.5
for i in range(3):
```

```
    if i>0:
        Signal_ch *=2
        Signal /=2
    print('Signal_ch',Signal_ch,'Signal',Signal,'alpha2',Signal_ch*Signal**2)

    SI = np.zeros((Size,Size,Channels))
    SI = build_dataset(SI,Signal_ch,Signal)
    SI = add_gaussian_noise(SI,0,Sigma2)
    denoised =make_pca(SI,1)[:,:,0]
    denoised = swop_comp(denoised)
    images.append(axs[1,i].imshow(denoised,vmin=-1,vmax=1))
    axs[1,i].set_axis_off()

plt.show()
```

Ivan:

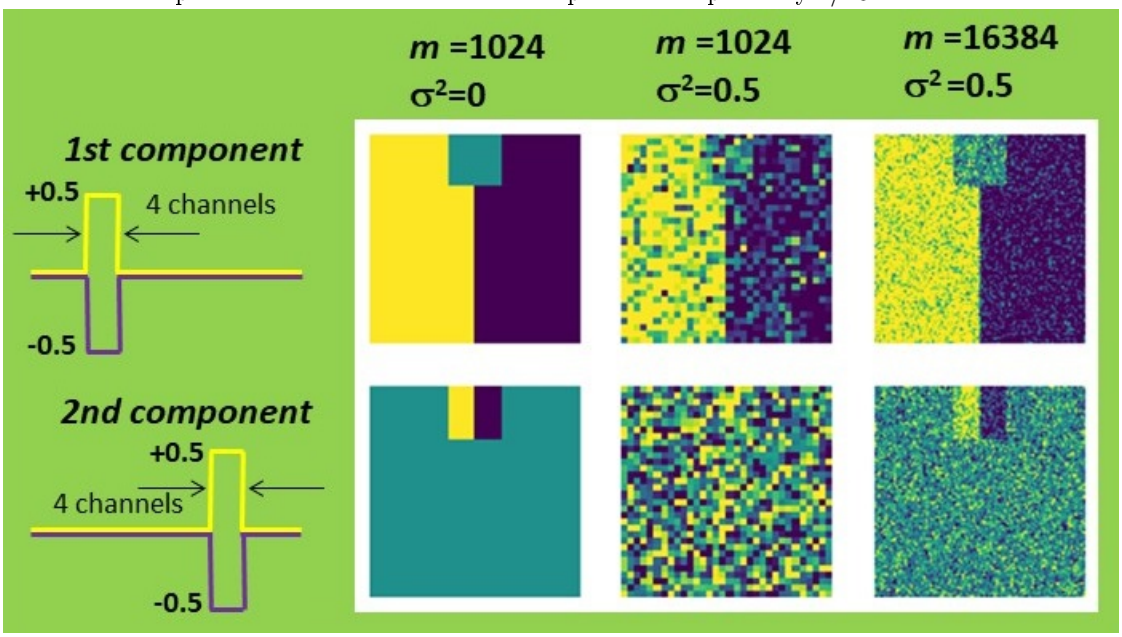
Thank you for the good example. However, I think the topic is not completely clarified. Suppose that we have an atomic lattice with two kinds of atoms, A and B. But at one cell, atom B is replaced for atom C. Assume that PCA sucessfully denoises A-B lattice but doesnot retrieve a singular atom C. The formula (1) in your post advises to increase the number of pixels to improve accuracy. But even if we scan over ten times more A-B cells, that would not help to uncover a singular C atom. This violates a common sense. Thus, I think the theory is incomplete and must be extended to the case of multicompound system and account for interaction among compounds.

Pavel:

The theory is complete, just its presentation in the post is fragmentary. You are right, in multi-component sets, the things can be a bit more tricky.

To model the situation you described, I introduced a small 8x8 pixels fragment into the 32x32 pixel set. In this fragment, quite different spectroscopy channels are activated, as if another chemical element appears or disappears. PCA is expected to detect the second principal component, which varies within this small fragment only. This is indeed the case in the noise-free case.

Figure 36: Two-component dataset when the 2nd component occupies only 1/16 fraction of the whole area.

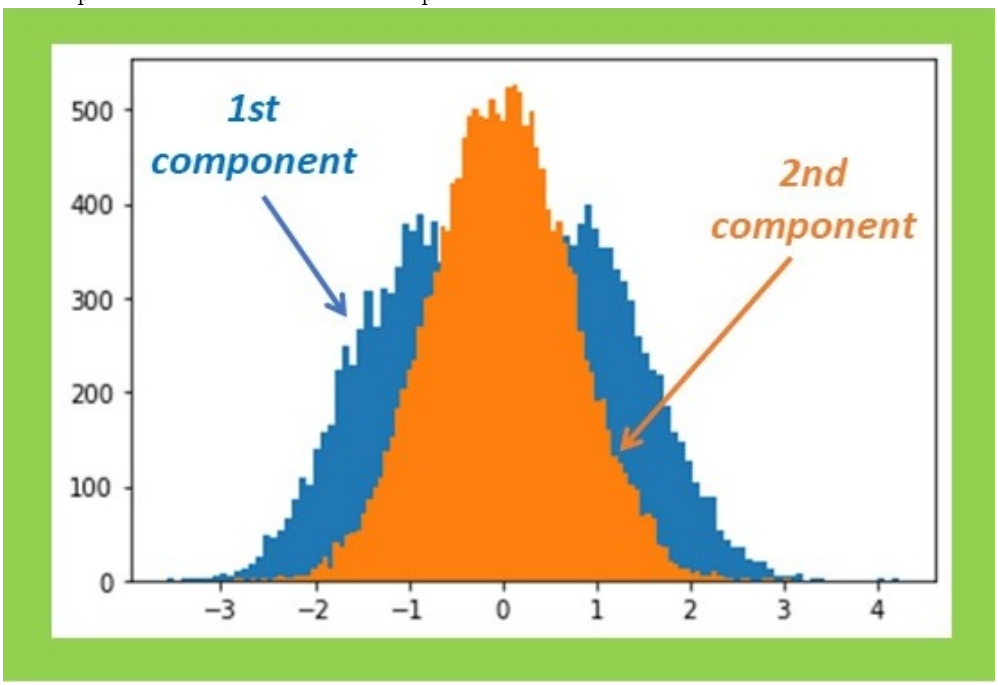


However, when noise is added, the second component suddenly becomes irretrievable. Why we observe the first component but not the second one? At the first glance, the parameters of formula (1) have not changed - added signal is of the same strength, the total number of pixel has not changed, and the noise level is same as for the first component.

Upon careful examination, it is revealed that the noise-free data distribution in the second component differs drastically from that in the first one. This is because the second component counts to zero in most

pixels, namely $1024 - 64 = 960$ pixels. Therefore, although the signal strength is same in both components, the data variance α^2 in the second component would be $1024/16 = 16$ less than that in the first one. This easy to notice if we recall that the variance is the *average* squared deviation from the mean (zero in this case). According to formula (2) the second component would reach the Nadler threshold at 16 times lower noise variance σ^2 than the first component.

Figure 37: The data distribution in the 2nd component is much sharper than that in the 1st one. This is because 2nd component is zero in most of the pixels.



You can verify this by examining the figure where σ^2 of 0.5 still appears acceptable for uncovering the variation of the first component but fully suppresses extraction of the second component. This limitation cannot be overcome by scanning over the larger area. In that case, m does increase but the variance of the second component decreases proportionally.

What would genuinely help is an increase in m through more dense scanning. The last column in the figure represents the same dataset with sampling increased four times. You can observe that both the first and second components are successfully retrieved now. While m increases by 16 times, α^2 remains the same as the number of pixels in the fragment increases proportionally. It is easy to confirm with formula (2) that the second component is now below the Nadler threshold.

Listing 10: Extra standard functions (functions2).

```
import numpy as np

#consruct SI dataset with zero mean and second component varying in small fragment
def build_dataset_2comp(SI,Signal_ch,Signal,Components_ratio):
    Size = SI.shape[0]
    Channels = SI.shape[2]
    Middle =int(Size/2)
    Smaller_cell =int(Size*Components_ratio)
    Half_cell = int(Smaller_cell/2)

    #variation of 1st component
    Channel_I1 =int(Channels/2)
    Channel_F1 =Channel_I1 + Signal_ch
    SI[:,:Middle,Channel_I1:Channel_F1] =Signal
    SI[:,Middle:Size,Channel_I1:Channel_F1] =-Signal

    #variation of 2nd component
```

```
    Channel_I2 =int(Channels/4)
    Channel_F2 =Channel_I2 + Signal_ch
    SI[:Smaller_cell,Middle-Half_cell:Middle,Channel_I2:Channel_F2] =Signal
    SI[:Smaller_cell,Middle:Middle+Half_cell,Channel_I2:Channel_F2] =-Signal

    #remove 1st component from the small cell
    SI[:Smaller_cell,Middle-Half_cell:Middle+Half_cell,Channel_I1:Channel_F1] =0

    return SI

#simpler swop of two components
def swop_comp2(denoised):
    Width =denoised.shape[1]
    left_half =denoised[:,:int(Width/2)].copy()
    if np.mean(left_half) <0:
        denoised[:,:int(Width/2)] *=(-1)
        denoised[:,int(Width/2):Width] *=(-1)
    return denoised
```

Listing 11: PCA-reconstructed maps of two-component data set where the second component is spatially strongly localised.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *
from functions2 import *


Pixels =1024
Channels =64
Size= int(m.sqrt(Pixels))
Signal_ch =4
Signal =0.5
Sigma2 =0.5

fig, axs = plt.subplots(2,3)
#fig.subplots_adjust(left=0.25,right=0.75)
images = []

SI = np.zeros((Size,Size,Channels))
SI = build_dataset_2comp(SI,Signal_ch,Signal,0.25)

#pca of noise-free set
denoised_2comp = make_pca(SI,2)
for k in range(2):
    denoised = denoised_2comp[:,:,k]
    print('noise-free set: variance of pca component',k+1,':',round(np.var(denoised),4))
    images.append(axs[k,0].imshow(denoised,vmin=-1,vmax=1))
    axs[k,0].set_axis_off()


#pca of noisy set
SI.shape =(Size,Size,Channels)
SI = add_gaussian_noise(SI,0,Sigma2)
denoised_2comp = make_pca(SI,2)
for k in range(2):
    denoised = denoised_2comp[:,:,k]
    if k==0:
        denoised[:int(Size/4),int(Size*3/8):int(Size*5/8)] =swop_comp2(denoised[:int(Size/4),int(
            Size*3/8):int(Size*5/8)])
    if k==1:
        denoised =swop_comp2(denoised)
```

```
    images.append(axs[k,1].imshow(denoised,vmin=-1,vmax=1))
    axs[k,1].set_axis_off()

#pca of noisy set with increased sampling
Pixels = 16384
Size= int(m.sqrt(Pixels))
SI = np.zeros((Size,Size,Channels))
SI = build_dataset_2comp(SI,Signal_ch,Signal,0.25)

SI.shape =(Size,Size,Channels)
SI = add_gaussian_noise(SI,0,Sigma2)
denoised_2comp = make_pca(SI,2)
for k in range(2):
    denoised = denoised_2comp[:,:,k]
    if k==0:
        denoised[:int(Size/4),int(Size*3/8):int(Size*5/8)] =swop_comp2(denoised[:int(Size/4),int(
            Size*3/8):int(Size*5/8)])
    if k==1:
        denoised =swop_comp2(denoised)
    images.append(axs[k,2].imshow(denoised,vmin=-1,vmax=1))
    axs[k,2].set_axis_off()

plt.show()

#histograms of noise-free components
bins=100
plt.hist(denoised_2comp[:,:,0].flatten(),bins=bins)
plt.hist(denoised_2comp[:,:,1].flatten(),bins=bins)
plt.show()
```

Ivan:

There is also so called "local PCA" to deal with such an issue. Can you comment ?

Pavel:

Yes. The "local PCA" introduced by Ishizuka and Watanabe in the conference in Prague in 2014 was designed exactly for the described situation when some components are expected to be strongly spatially localised. The authors suggested to break the whole dataset on equal fragments, like a grid and perform PCA in each fragment independently.

Let's apply this strategy to our example and divide the set into smaller fragment. Please forgive me for cutting a bit the edges of the set, otherwise programming would be too complicated. You see from figure below that the local PCA indeed precludes the loss of the second component but makes the first component more noisy. I will try to explain why it happens.

At *small* σ^2 the accuracy of the second component extraction is not changed by local PCA. Indeed, m is decreased 16 times while α^2 is 16 time increased, thus formula (1) remains balanced. However, the situation is different at *high* σ^2 when formula (2) should be applied. It is easy to see that the right part of (2) decreases slower with m when comparing to the left one. As a result, the larger noise level σ^2 is needed to reach the Nadler threshold for the loss of information.

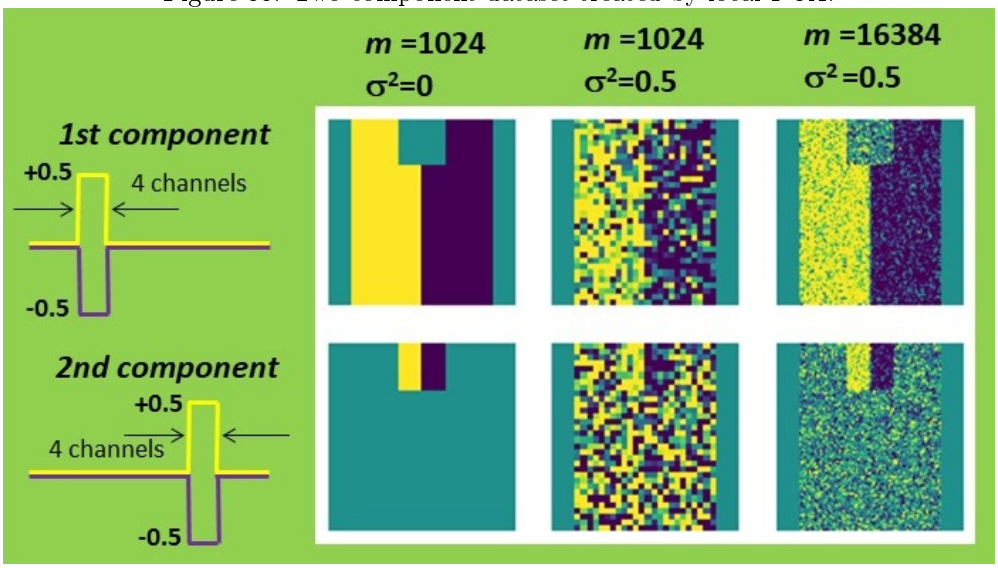
Such strategy is however not good for the dominant first component. The local PCA does not profit from averaging over the large area and the first component is more affected by noise. This peculiarity of local PCA is highlighted in the Table below.

	1st component	2nd component
PCA	$\sigma^2 < 4$	$\sigma^2 < 1/4$
local PCA	$\sigma^2 < 1$	$\sigma^2 < 1$

Table 1: Requirement to preserve information for a given example. Classical PCA preserves the 1st component till the noise level 4 while dissolves the second component already at the level $1/4$. The local PCA equalizes the chances for both components.

To summarize: the local PCA can be useful however requires a great care - it improves one things while worsening the others. I would recommend the following:

Figure 38: Two-component dataset treated by local PCA.



1. Apply it only when datasets consist of periodic fragments like atomic lattice and you have a strong suspicion that the unit cells are not identical.
2. Set the PCA fragments approximately equal to the unit cell size. The smaller size would add more noise to the PCA results.
3. Always compare to the PCA of the whole set.

Listing 12: Maps of two-component data set reconstruction with local PCA.

```
"""
INSTALL FIRST MODULE CONSISTING ALL TYPICAL FUNCTIONS'
"""
from functions import *
from functions2 import *

def loop_pca_vertically(SI,denoised_2comp,IniX,FinX,Subsize):
    for j in range(4):
        IniY = int(j*Subsize)
        FinY = int((j+1)*Subsize)
        subSI =SI[IniY:FinY,IniX:FinX,:].copy()
        sub_denoised_2comp = make_pca(subSI,2)

        for k in range(2):
                denoised_fragm =sub_denoised_2comp[:,:,k]
                denoised_fragm =swop_comp2(denoised_fragm)
                denoised_2comp[IniY:FinY,IniX:FinX,k] = denoised_fragm

    return denoised_2comp

def treat_locally(SI):
    Size =SI.shape[0]
    denoised_2comp =np.zeros((Size,Size,2))
    Subsize =int(Size/4)

    #central row
    IniX =int(3/8*Size)
    FinX =int(5/8*Size)
    denoised_2comp = loop_pca_vertically(SI,denoised_2comp,IniX,FinX,Subsize)
    #retrieved component in this fragment is actually the 2nd one, swop it:
    buffer = denoised_2comp[:Subsize,IniX:FinX,0].copy()
    denoised_2comp[:Subsize,IniX:FinX,0] = denoised_2comp[:Subsize,IniX:FinX,1]
```

```
    denoised_2comp[:Subsize,IniX:FinX,1] = buffer

    #left row
    IniX =int(1/8*Size)
    FinX =int(3/8*Size)
    denoised_2comp = loop_pca_vertically(SI,denoised_2comp,IniX,FinX,Subsize)
    #add the average
    denoised_2comp[:,IniX:FinX,0] +=1

    #right row
    IniX =int(5/8*Size)
    FinX =int(7/8*Size)
    denoised_2comp = loop_pca_vertically(SI,denoised_2comp,IniX,FinX,Subsize)
    #add the average
    denoised_2comp[:,IniX:FinX,0] -=1

    return denoised_2comp


Pixels =1024
Channels =64
Size = int(m.sqrt(Pixels))
Signal_ch =4
Signal =0.5
Sigma2 =0.5

fig, axs = plt.subplots(2,3)
images = []


#local pca of noise-free set with 1024 pixels
SI = np.zeros((Size,Size,Channels))
SI = build_dataset_2comp(SI,Signal_ch,Signal,0.25)
denoised_2comp =treat_locally(SI)

for k in range(2):
    images.append(axs[k,0].imshow(denoised_2comp[:,:,k],vmin=-1,vmax=1))
    axs[k,0].set_axis_off()
print('noise-free set: variance:',round(np.var(denoised_2comp[:4,12:20,1]),4))


#local pca of noisy set with 1024 pixels
SI = np.zeros((Size,Size,Channels))
SI = build_dataset_2comp(SI,Signal_ch,Signal,0.25)
SI = add_gaussian_noise(SI,0,Sigma2)
denoised_2comp =treat_locally(SI)
for k in range(2):
    images.append(axs[k,1].imshow(denoised_2comp[:,:,k],vmin=-1,vmax=1))
    axs[k,1].set_axis_off()

#local pca of noisy set with 16384 pixels
Pixels =16384
Size = int(m.sqrt(Pixels))

SI = np.zeros((Size,Size,Channels))
SI = build_dataset_2comp(SI,Signal_ch,Signal,0.25)
SI = add_gaussian_noise(SI,0,Sigma2)
denoised_2comp =treat_locally(SI)
for k in range(2):
    images.append(axs[k,2].imshow(denoised_2comp[:,:,k],vmin=-1,vmax=1))
    axs[k,2].set_axis_off()
```

```
plt.show()
```

5 Squeezing dimensions

5.1 James Bond tells the story

I've always admired James Bond's knack for wriggling out of impossible situations. I said to him:

"It's quite remarkable how smoothly you scale fences, leap from windows, and bulldoze through walls. I find myself wishing I could be a bit more like you..."

Figure 39: You wish to experience claustrophobia?

James' response, however, was less than encouraging.

"You wish to experience claustrophobia?" he retorted, arching an eyebrow.

"You mean to tell me you're uncomfortable in a cramped cage with tied hands?"

"Exactly," He admitted, nodding. "And in an elevator cabin with closed doors too. I've never been a fan of those maze attractions either. Whenever faced with a labyrinth, I simply opt for a simple ladder and make my escape into the third dimension."

"But what if your enemies manage to trap you in a cell with a closed roof and floor?" I countered.

"Same strategy," replied he confidently. "I'll find my way out through an extra dimension."

"But we live in a three-dimensional space, you know."

"Are you sure?" James smirked, shaking his head.

5.2 Are we living in 3 dimensions?

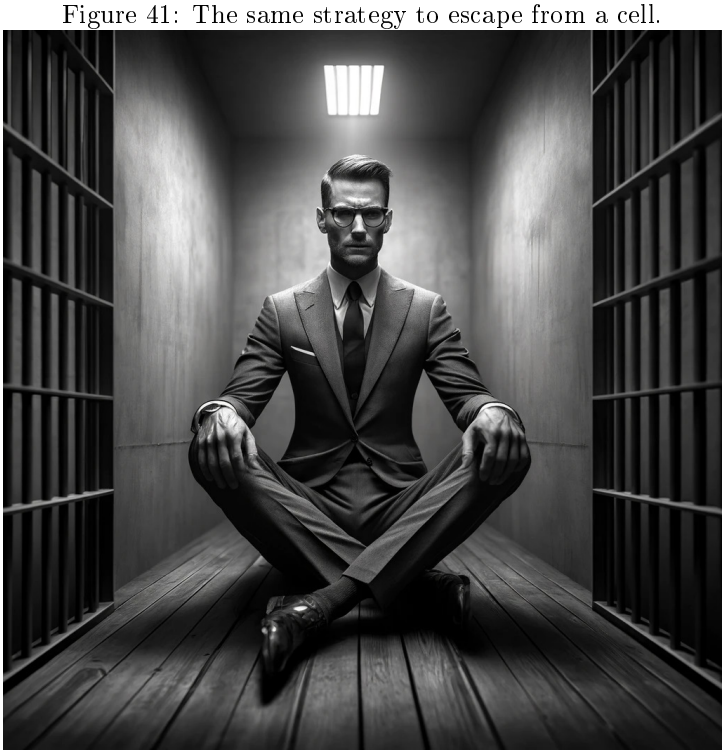
Bond was onto something. The true dimensionality of our world remains a mystery. It seems our brains have settled on encoding it as a three-dimensional space, but this choice is purely pragmatic. Throughout our evolutionary history, activities in other dimensions didn't offer any survival advantages, so our brains streamlined their processing to focus on the three dimensions most relevant to our daily lives.

It's likely that our neural networks somehow compress the external reality to achieve this reduction in dimensionality. And no, I'm not referring to the time dimension introduced most notably by Albert Einstein (our brains haven't quite grasped that one yet, by the way). There could be other dimensions lurking beyond our perception, but since they don't offer practical utility in the vast majority of cases, we remain oblivious to them.

So, just like Principal Component Analysis (PCA) condenses multi-dimensional data by selecting a few major components and discarding the rest, our minds perform a similar feat. By truncating the world to three dimensions, we simplify the cognitive load, making it easier to navigate and comprehend. It's a fascinating parallel: our mental autoencoder and PCA both strive to reduce complexity, enabling us to operate more efficiently within our dimensional framework.

Figure 40: James' method to get out of a maze.

Figure 41: The same strategy to escape from a cell.

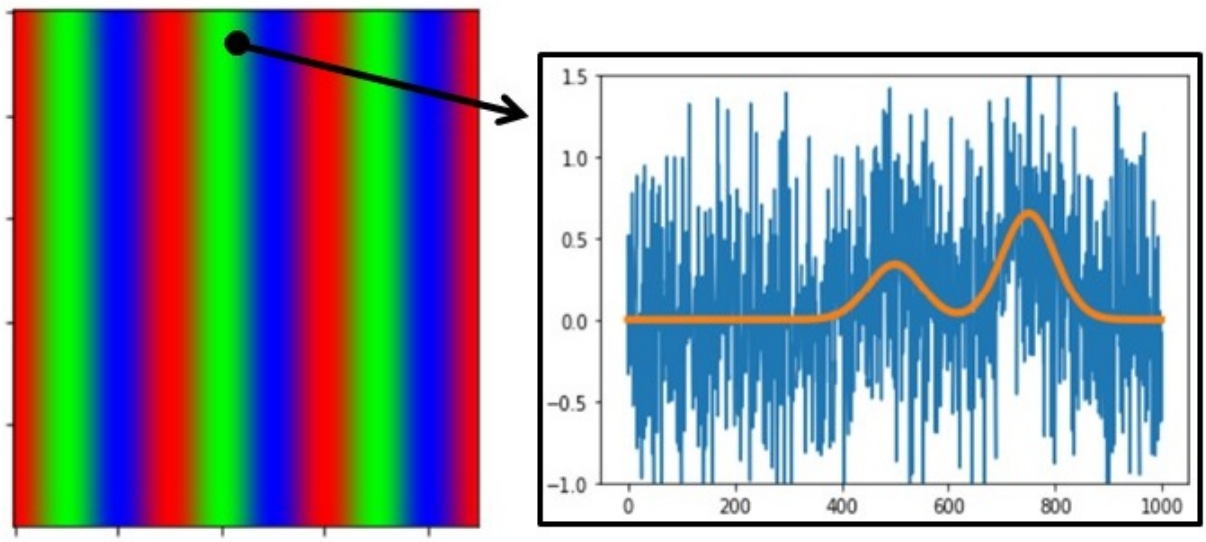


5.3 Best way to truncate the PCA dimensions

This truncation process lies at the heart of PCA. To illustrate that, I constructed a synthetic map featuring three compounds, each emitting distinct spectroscopic peaks. Adding Gaussian noise for realism (Poisson noise would have been more appropriate, but I opted for simplicity), I created a scenario where each pixel of the map could potentially emit a continuous or discrete signal across 1000 energy channels. This translates to a staggering 1000 dimensions in our data space, making navigation cumbersome.

Figure 42: Are you sure that we live in a three-dimensional space?

Figure 43: Layered structure consisting of 3 compounds where a spectrum from each pixel is taken.



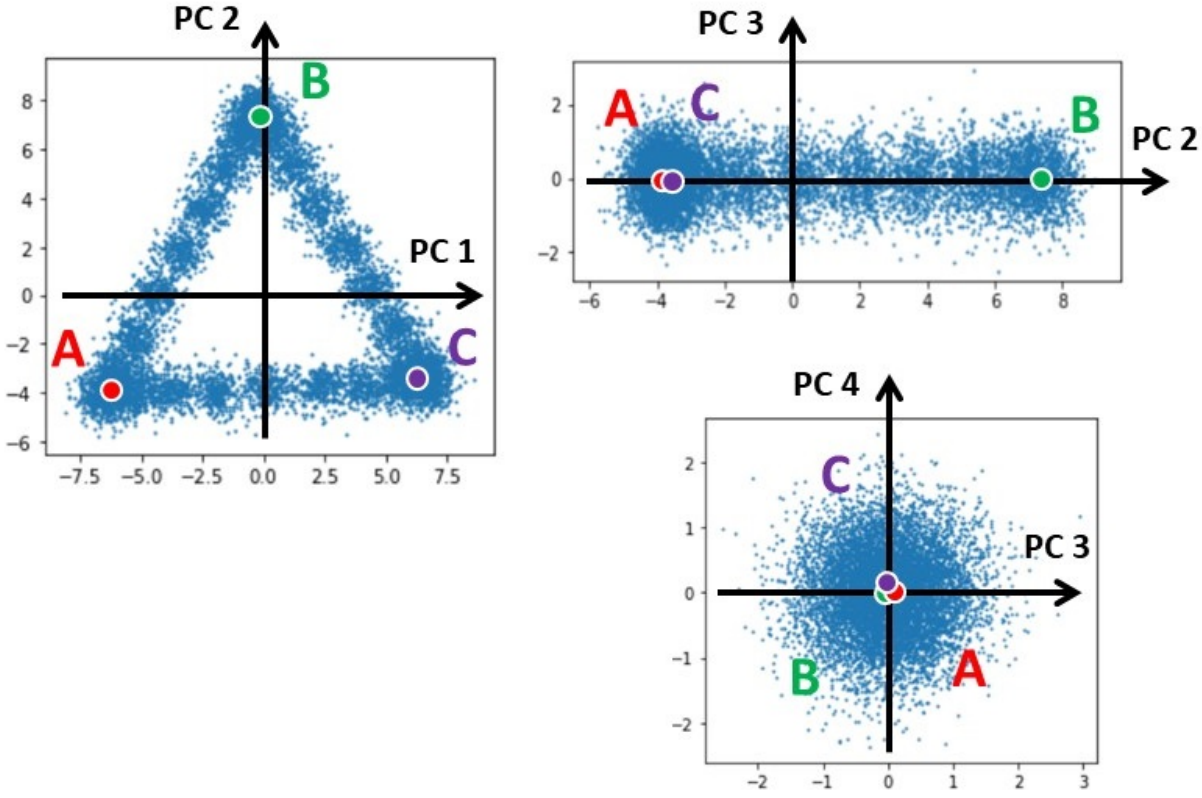
Enter PCA. By extracting, for instance, the ten most significant directions (principal components) from this 1000-dimensional space, we can squeeze data points into the more manageable volume. Look at the two-dimensional projection of this volume, specifically at the plane formed by the first and second principal components. You see a clear delineation of the data points corresponding to compounds A, B, and C, allowing seamless navigation among them.

However, projecting onto the (second plus third) principal components plane reveals a lack of meaningful variation along the third axis, indicating a mere Gaussian spread of noise. Similarly, projecting onto the (third plus fourth) plane yields a two-dimensional Gaussian distribution devoid of material information, serving only to quantify noise levels.

Thus, what initially seemed like a daunting 1000-dimensional dataset reveals itself to be effectively two-dimensional. Even the ten principal components we initially extracted appear excessive.

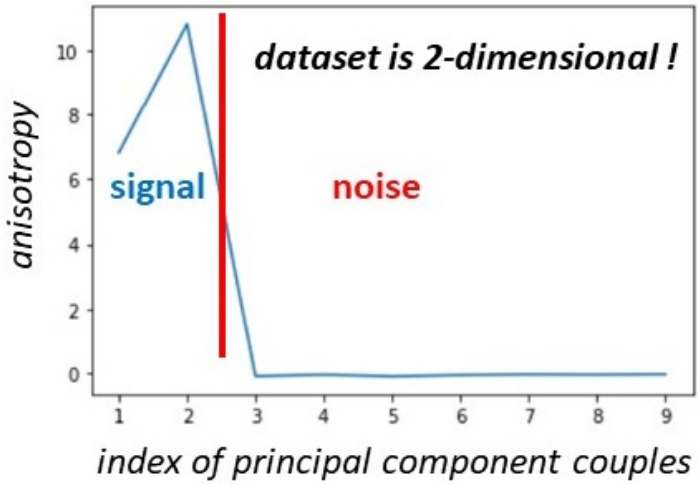
Still, accurate determining the dataset's true dimensionality requires a more nuanced approach. Let's try to estimate a kind of anisotropy of these two-dimensional projections. Say, to measure how differently data are distributed along the randomly chosen directions. Such anisotropy parameter should be zero for

Figure 44: Data distribution projected on planes formed by different principal components.



directionally uniform distributions and non-zero for anisotropic ones. The possible Python implementation can be found below.

Figure 45: Anisotropy of joint distribution of different principal components plotted in ascending order.



A straightforward computational solution emerges: identify and retain principal components couples exhibiting anisotropy, while discarding those that align isotropically. My approach is, of course, not the only one. You might look at the following alternatives: `https://tminka.github.io/papers/pca/` or `https://arxiv.org/abs/1311.0851`. Yet, as James Bond remarked, "It doesn't matter who and how, what matters is the mission is performed".

5.4 Used codes

Listing 13: Truncation of principal components.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.decomposition import PCA

def gaussian_signal(Depth, mu, sig):
    x =np.arange(Depth)
    return np.exp(-np.power((x - mu)/sig, 2.) / 2)

def signal(D,Start,End,Fract,NoiseSigma):
    Sigmas =5 #+-sigma in range
    Sig =(End -Start)/2/Sigmas
    Mu = (Start +End)/2

    spec =np.arange(D)
    #spec = gaussian(spec,Mu,Sig)*Fract

    if NoiseSigma >0:
        spec += np.random.normal(0,NoiseSigma,D)

    return spec

def compound_layer(Height,Width):
    axis = np.arange(Width)
    profile = (1 - np.cos(2 * np.pi * axis /Width)) /2 #sinusoidal profile
    map_c =np.ones((Height,Width))
    map_c *=profile #sinusoidal distribution from left to right

    return map_c

def layers_fragment(Height,Width):
    fragment =np.zeros((Height,Width,3))
    HW =Width //3
    fragment[:,:HW,0] =compound_layer(Height,2*HW)[:,HW:] #right half of A
    fragment[:,:2*HW,1] =compound_layer(Height,2*HW)   # B compound
    fragment[:,HW:,2] =compound_layer(Height,2*HW)     # C compound
    fragment[:,2*HW:,0] =compound_layer(Height,2*HW)[:,:HW] #left half of A

    return fragment

def make_SI_3features(im,Depth,SignalSigma,NoiseSigma):
    Height =im.shape[0]
    Width =im.shape[1]
    imSI =np.zeros((Height,Width,Depth))

    for y in range(Height):
        for x in range(Width):
            #print(x,y)
            Feature_A = im[y,x,0]
            Feature_B = im[y,x,1]
            Feature_C = im[y,x,2]
            spec = Feature_A*gaussian_signal(Depth,Depth/4,SignalSigma) #add 1st feature
            spec += Feature_B*gaussian_signal(Depth,2*Depth/4,SignalSigma) #add 2nd
            spec += Feature_C*gaussian_signal(Depth,3*Depth/4,SignalSigma) #add 3rd
            if NoiseSigma >0: #add Gaussian noise
                spec += np.random.normal(0,NoiseSigma,Depth)
            imSI[y,x,:] =spec
```

```
    return imSI

Width =90
Height=100
maps =np.zeros((Height,Width,3))
for i in range(3): maps[:,i*(Width//3):(i+1)*(Width//3),:] = layers_fragment(100,30)

plt.imshow(Image.fromarray((255*maps).astype('uint8')))
plt.show()

Depth =1000
SignalSigma=50
NoiseSigma=0.5
SI = make_SI_3features(maps,Depth,SignalSigma,NoiseSigma)
spec =SI[50,46,:]
spec.shape =(Depth,)
plt.plot(np.arange(Depth),spec)
plt.show()

Matrix = SI.copy()
Matrix.shape =(Height*Width,Depth)

Extracted_components =10
pca = PCA(n_components=Extracted_components)
pca.fit(Matrix)
scores =pca.transform(Matrix)
print(scores.shape)

def scatterplot(scores,First,Second):
    plt.scatter(scores[:,First-1],scores[:,Second-1],s=1)
    ax = plt.gca()
    ax.set_aspect('equal')
    plt.show()

scatterplot(scores,1,2)

from math import pi

def scatterLimits(score,Limit:float)->tuple:
    #plain min and max
    Mini =np.min(score)
    Maxi =np.max(score)

    #squise data within a predefined fraction of standard deviation
    if Limit !=None:
        Mean =np.mean(score)
        StDev =np.std(score)
        Mini =max(Mean-StDev*Limit,Mini)
        Maxi =min(Mean+StDev*Limit,Maxi)

    return Mini,Maxi

def rotVectors_0_90(Orients:int) ->list:
    vec =np.zeros((2,Orients))
    X =np.arange(Orients)
    vec[0,:] =np.cos(X*pi/(Orients-1)/2)
    vec[1,:] =np.sin(X*pi/(Orients-1)/2)

    return vec


def anisotropy(scores,First:int,Second=None, Whitening =False,AniParameters=None):
```

```
    if AniParameters==None:
        Cell =20
        Limit =3
        Rots =18
    else:
        Cell =AniParameters[0]
        Limit =AniParameters[1]
        Rots =AniParameters[20]

    if Second ==None: #two consequent scores
        Second =First+1
    Length =scores.shape[0] #number of pixels
    score1 = scores[:,First:First+1]
    score2 = scores[:,Second:Second+1]
    score1.shape =(1,Length)
    score2.shape =(1,Length)

    #limits
    Mini1,Maxi1 =scatterLimits(score1,Limit)
    #print('Limit',Limit,'min',Mini1,'max',Maxi1)
    Scaling =1
    if Whitening ==True: #discard the difference in variance
        Mini2,Maxi2 =scatterLimits(score2,Limit)
        Scaling =(Maxi1/Maxi2 + Mini1/Mini2)/2 #scale approximately same deviations from zero

    Bins =int(Length/Cell) #number of bins in histogram
    #such as a given number of points (Cell) fall into one pixel
    #(at plain distribution)

    vecRotated =rotVectors_0_90(Rots+1)

    coupleScores =np.zeros((Length,2))
    coupleScores[:,0] =score1
    coupleScores[:,1] =score2*Scaling
    projections =np.dot(coupleScores,vecRotated) #projections to series of unit vectors

    hist2D =np.zeros((Rots+1,Bins)) #histograms for all projections
    for i in range(Rots+1):
        hist2D[i,:] =np.histogram(projections[:,i], range=(Mini1,Maxi1), bins=Bins)[0]

    histMean =hist2D.mean(0) #mean histogram
    hist2D -=histMean #deviations from mean
    hist2D =np.square(hist2D) #squared deviations

    with np.errstate(divide='ignore', invalid='ignore'):
        hist1D = np.true_divide(hist2D,histMean) #normalize on counts
        hist1D[hist1D == np.inf] = 0
        hist1D = np.nan_to_num(hist1D)

    Ani =hist1D.sum() #sum of squared deviations
    Ani /=Bins        #normalize on Bins
    Ani /=(Rots+1)    #normalize on rotations number
    Ani -=1           #criterion -> ZERO

    return Ani


def anisotropy_plot(scores,Whitening =False,AniParameters=None):
    Pairs =scores.shape[1]-1 #number of couple is one less than number of scores

    plot =np.zeros(Pairs)
    for i in range(Pairs):
```

```
        Anisotropy =anisotropy(scores,i,Whitening =Whitening,AniParameters=AniParameters)
        #print(i+1,Anisotropy)
        plot[i] =Anisotropy

    return plot

aniso_plot = anisotropy_plot(scores)
plt.plot(np.arange(Extracted_components-1)+1,aniso_plot)
plt.show()
```

John:

You state that there are other dimensions we do not see usually. Do they consist of noise like PCA minor components?

Pavel:

I did not state anything, it was just the (most probably inaccurate) speculations. PCA selects the most relevant dimensions and rejects the rest as the rest appears to be noise in most cases. Why not our brains do the similar job? Is that exactly noise or something else what we rejected? I don't know. We only can say that seeing this extra information did not help us and our predecessors to survive. Thus, the evolution allowed us to cognize only 3 conventional spatial dimensions plus time dimension.

John:

You think we perceive only three dimensions? Then it is hard to explain why for example, a certain sequence of sounds, music, affects us so much.

Pavel:

I feel there is a sense in your words, but I am not ready to support fully this idea. Let me think about that.

6 On the Merits of Indolence

6.1 James Bond tells the story

What was never clear to me about James Bond's personality was why he was dubbed 'agent 007.' I queried him once if there were at least six other super-agents comparable to him in skills, intelligence, and experience.

He responded, 'This nickname actually has another origin. My colleagues jest that I possess zero motivation, zero concentration, and seven romantic adventures per mission.'

Figure 46: James Bond on a mission.

'What unfairness!' I screamed. 'To judge so superficially... They should consider your excellent results...'

'I must confess,' remarked Bond, 'they were not entirely incorrect. I've never been particularly industrious, but I've endeavored to compensate for any deficiencies in acquired information through advanced analysis.'

'How thrilling!' I exclaimed. 'How do you manage that?'

'Ah, now we tread upon my most closely guarded professional secrets,' he replied, glancing about for prying eyes, listening devices, and surveillance cameras before hastily scribbling something on a small piece of paper.

'Top secret!' he cautioned as he slipped the paper into my pocket.

6.2 Gaussian Process

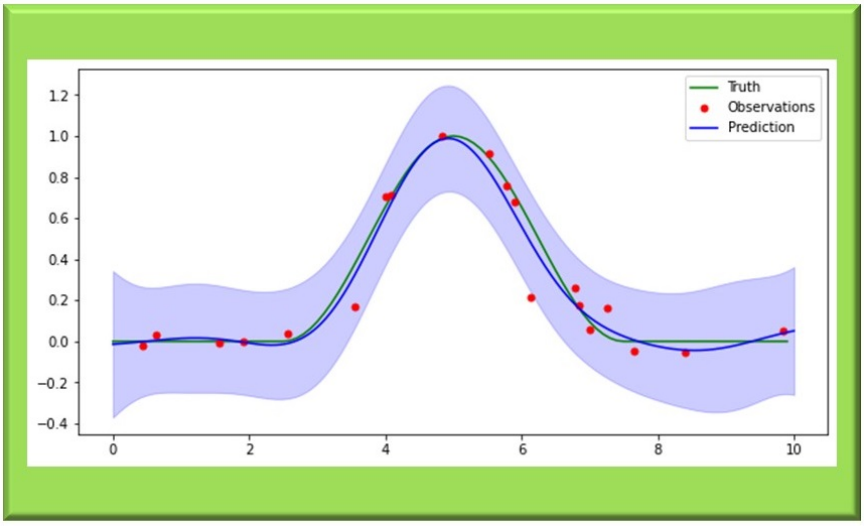
I could scarcely resist the urge to unfurl the paper immediately, yet I restrained myself until I reached home and secured all the lockers. Upon unfolding it, I read the words: 'Gaussian Process.'

'Of course!' I thought to myself. 'I should have deduced it on my own. The Gaussian Process, the most precise Bayesian prediction method for filling in the missing pieces of information.

Figure 47: Confidential keyword.

For instance, when tasked with retrieving an unknown 1D functional dependence, the initial inclination may be to sample it at equal intervals. However, this approach proves to be both costly and inefficient. Instead, a much more effective method involves random sampling, followed by probabilistic filling of the missing points using the Gaussian Process.

Figure 48: Few random sampling allow for accurate reconstruction of the unknown function with the Gaussian Process. The blue area around the predicted curve shows the confidence interval.



This strategy proves particularly efficacious for retrieving 2D features with a limited sampling budget. I recently conducted an experiment wherein I generated a cosine blob at the center of an image and sampled it with only 100 randomly chosen points. And the Gaussian Process reconstructed the true feature with remarkable accuracy. To provide a point of comparison, I also plotted (on the rightmost side) the reconstruction obtained from standard regular sampling, which appeared significantly less impressive despite employing 100x100 (=10,000) sampling points.

Figure 49: Two-dimensional feature randomly sampled followed by reconstruction with the Gaussian Process. This is very economical and more efficient than the regular sampling reconstruction.

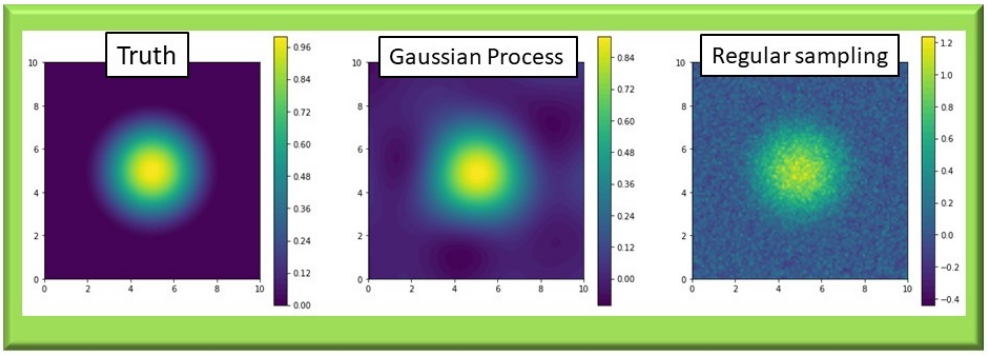
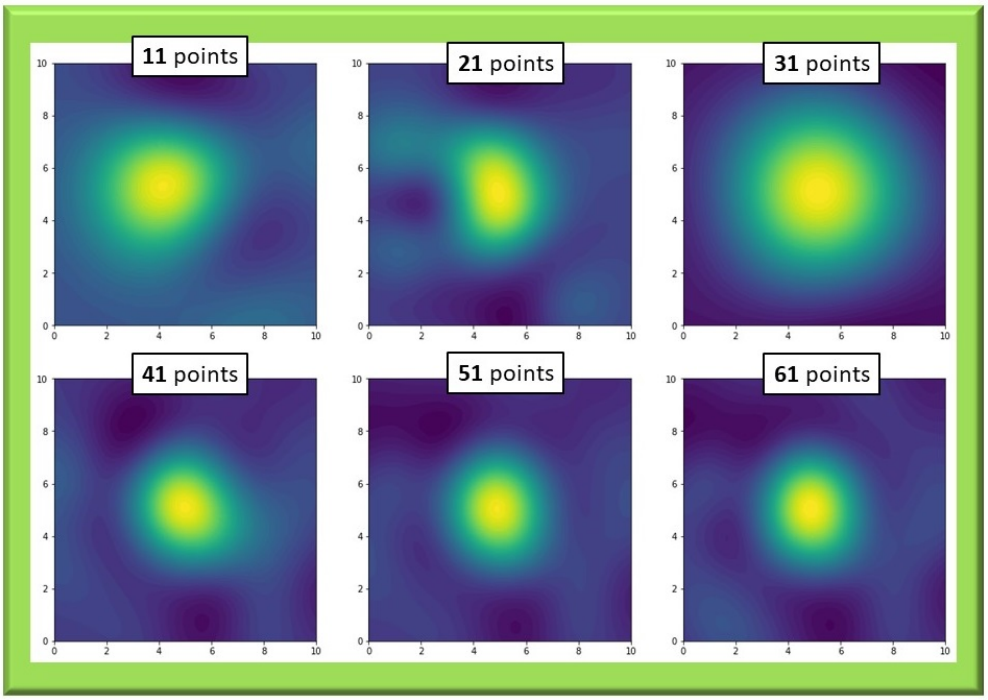


Figure 50: Evolution of Gaussian Process reconstruction with increasing of the number of sampling points.



Furthermore, it is intriguing to observe how the reconstruction evolves with the sequential addition of sampling points. The resulting image consistently maintains a smooth appearance but remains rather inaccurate when only a small number of points are taken. However, accuracy improves rapidly with the accumulation of more sampling points, approaching the original image closely even with just 50 samplings.

6.3 Used codes

Listing 14: Random sampling of a 1D function followed by the Gaussian Process reconstruction.

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C, WhiteKernel
import numpy as np
import matplotlib.pyplot as plt
import math as m

def feature(X,Width):
    func = (1+np.cos(X*4*m.pi/Width))/2
```

```
    func =np.where(X<Width/4,0,func)
    func =np.where(X>Width*3/4,0,func)
    return func

def generate_random_grid(Points,Width):
    rng = np.random.default_rng()
    #random values between 0 and Width
    X = np.sort(rng.uniform(0, Width, Points)).reshape(-1, 1)
    #calcalte feature at these points and add noise
    y = feature(X,Width).ravel() + rng.normal(0, 0.1, X.shape[0])
    return X, y

def train_gp(X, y):
    # Kernel: combination of a constant kernel, RBF kernel and WhiteNoise
    kernel = C(1.0, (1e-4, 1e0)) * RBF(length_scale=1.0,
           length_scale_bounds=(1e-2, 1e1))+ WhiteKernel(noise_level=0.1,
                                       noise_level_bounds=(1e-3, 1e1))
    gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)
    gp.fit(X, y)
    print(gp.kernel_)
    return gp


width =10
Points_reg =1000
Points =20

# Generate random grid
X_rnd, y_rnd = generate_random_grid(Points,width)

# Train Gaussian Process
gp = train_gp(X_rnd, y_rnd)

# Predict regular grid
X_reg = np.linspace(0, width, Points_reg).reshape(-1, 1)
y_pred, sigma = gp.predict(X_reg, return_std=True)

# Visualize
plt.figure(figsize=(10, 5))
plt.plot(np.arange(100)/width, feature(np.arange(100)/width,width), 'g-', label='Truth')
plt.plot(X_rnd, y_rnd, 'r.', markersize=10, label='Observations')
plt.plot(X_reg, y_pred, 'b-', label='Prediction')
plt.legend()

# Show confidence interval based on prediction spread sigma
plt.fill_between(X_reg.ravel(), y_pred - 1.96 * sigma, y_pred + 1.96 * sigma,
                 alpha=0.2, color='blue')
plt.show()
```

Listing 15: Random sampling of a 2D function followed by the Gaussian Process reconstruction.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C,WhiteKernel
import math as m


def feature(X, Y, width):
    R = np.sqrt((X - width / 2) ** 2 + (Y - width / 2) ** 2)
    func = (1 + np.cos(R * 3 * np.pi / width)) / 2
    func =np.where(R>width/3,0,func)
```

```
    return func

def generate_regular_grid(width, Samples,noise =0):
    x = np.linspace(0, width, Samples)
    y = np.linspace(0, width, Samples)
    X, Y = np.meshgrid(x, y)
    Z = feature(X, Y, width) + np.random.normal(0, noise, X.shape) # Adding some noise
    return X, Y, Z

def generate_random_grid(Width,Points,noise =0):
    rng = np.random.default_rng()
    X = rng.uniform(0, Width, Points).reshape(-1, 1)
    Y = rng.uniform(0, Width, Points).reshape(-1, 1)
    Z = feature(X, Y, width) + np.random.normal(0, noise, X.shape) # Adding some noise
    return X, Y, Z

def make_2d_grid(X,Y):
    # Flatten the matrices for fitting
    X_flat = X.ravel().reshape(-1, 1)
    Y_flat = Y.ravel().reshape(-1, 1)
    XY = np.hstack((X_flat, Y_flat))
    return XY

def train_gp_2d(X, Y, Z):
    XY =make_2d_grid(X,Y)
    Z_flat = Z.ravel()
    # Define and fit the Gaussian Process
    kernel = C(1.0, (1e-4, 1e1)) * RBF(length_scale=1.0) + WhiteKernel(noise_level=0.05)
    gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)
    gp.fit(XY, Z_flat)
    print(gp.kernel_)
    return gp

def subplot_contour(X,Y,Z,ind,Title):
    plt.subplot(1, 3, ind)
    plt.contourf(X, Y, Z, levels=50, cmap="viridis")
    plt.colorbar()
    plt.title(Title)
    plt.gca().set_aspect('equal')


width =10
Points_reg =10000
Samples =int(m.sqrt(Points_reg))
Points =100

X, Y, Z_true = generate_regular_grid(width, Samples)
X, Y, Z_reg = generate_regular_grid(width, Samples,noise=0.1)
X_rnd, Y_rnd, Z_rnd = generate_random_grid(width, Points,noise=0.1)

gp =train_gp_2d(X_rnd, Y_rnd, Z_rnd)
XY =make_2d_grid(X, Y)
Z_pred = gp.predict(XY).reshape(Samples, Samples)

plt.figure(figsize=(20, 6))
subplot_contour(X,Y,Z_true,1,"Truth")
subplot_contour(X,Y,Z_pred,2,"GP random")
subplot_contour(X,Y,Z_reg,3,"Regular")
```

Listing 16: Evolution of the Gaussian Process reconstruction with increasing sampling.

```
#### ADD FUNCTIONS FROM THE PREVIOUS LISTING ####
```

```
def subplot_contour(X,Y,Z,ind,Title):
    plt.subplot(2, 3, ind)
    plt.contourf(X, Y, Z, levels=50, cmap="viridis")
    plt.title(Title)
    plt.gca().set_aspect('equal')
    plt.colorbar().remove()

def add_random_points(X_rnd, Y_rnd, Z_rnd, Points):
    #print(X_rnd.shape)
    X_new,Y_new,Z_new =generate_random_grid(width, Points,noise=0.1)
    X_rnd =np.concatenate((X_rnd,X_new),axis=0)
    Y_rnd =np.concatenate((Y_rnd,Y_new),axis=0)
    Z_rnd =np.concatenate((Z_rnd,Z_new),axis=0)

    gp =train_gp_2d(X_rnd, Y_rnd, Z_rnd)
    return X_rnd, Y_rnd, Z_rnd, gp

width =10
Added_points =10
Samples =100

X, Y, Z_true = generate_regular_grid(width, Samples)
XY =make_2d_grid(X, Y)

X_rnd =np.zeros((1,1))
Y_rnd =np.zeros((1,1))
Z_rnd =np.zeros((1,1))

plt.figure(figsize=(18, 12))
for i in range(6):
    X_rnd, Y_rnd, Z_rnd, gp =add_random_points(X_rnd, Y_rnd, Z_rnd,Added_points)
    Z_pred = gp.predict(XY).reshape(Samples, Samples)
    Points = X_rnd.shape[0]
    print('points',Points)
    subplot_contour(X,Y,Z_pred,i+1,Points)
```

Michael:

Did you hear about compressed sensing?

Pavel:

Yes, I did. Actually, this story is exactly about compressed sensing, although the reconstruction algorithms may differ. L1 sparsity algorithms are more common in compressed sensing, e.g. `https://arxiv.org/abs/1211.5231`. However, I think the Gaussian Process is more elegant.

7 Art of prophesy

7.1 James Bond tells the story

One of the most exhilarating scenes in spy movies is the daring escape of a secret agent amidst a hailstorm of enemy gunfire. A cunning agent avoids a direct path, instead weaving through a serpentine course that makes targeting him a daunting task for his adversaries, who struggle to anticipate his next move.

Figure 51: It is hard to predict a move of a cunning secret agent.

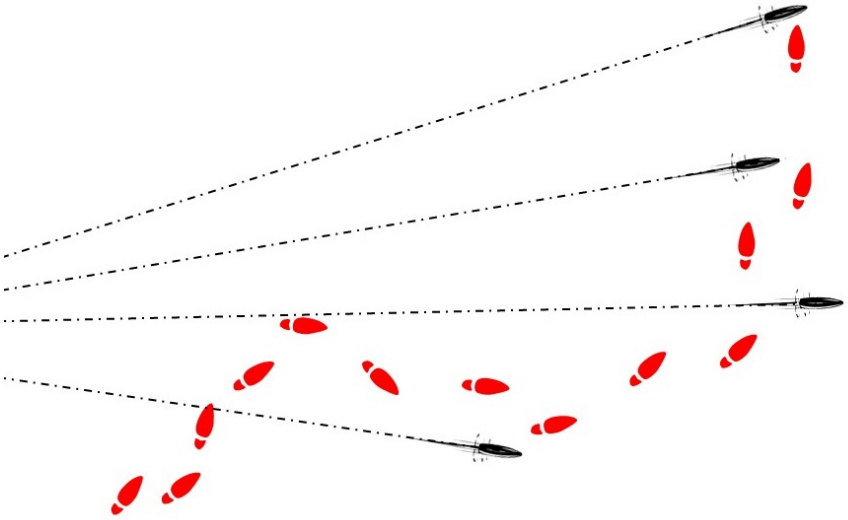


Figure 52: Unforeseen situation.

James interjected, "It's not very common, although, once I appeared exactly in the situation you described"

"Was it a sniper from a foreign agency?" I inquired.

A shadow crossed the Bond' face.

"Actually, it was a jealous husband. That was a case where I failed to foresee the circumstances quite accurately..." Bond confessed, before firmly pushing aside the unpleasant memories and continuing,

"But it's not important. In most instances, our aim is to predict not the trajectory of bullets, but rather the intentions and even the mental states of our adversaries. We employ various models and artificial neural networks."

"And do they accurately predict the future?"

James maintained the optimistic facade but with some hints of doubts at his face.

"You know, it is very difficult to make predictions, especially about the future... "

Figure 53: The powerful MI-6 artificial neuronal networks can predict the result of a coin toss experiment with the precision of up to 50%.

7.2 Predicting time series with LSTM networks

Can one become a sort of oracle by employing the hints of James Bond? I pondered this question as I embarked on a simple experiment. Constructing a basic function, say a sinusoidal wave, I posed the question:

"If we observe such an oscillating time series for an extended time, can we forecast its future behavior?"

To increase the challenge, I introduced noise into the equation, ensuring that extrapolation alone could not decipher the pattern.

I devised a rudimentary neural network with a couple of LSTM layers, augmented by a dropout layer to prevent overfitting. Configuring the model to utilize the preceding 50 measurements to predict the subsequent value, I discovered that a network trained on 70% of the complete time series could reasonably extrapolate the remaining 30% of data.

However, there is honestly a kind of cheating in this figure. The network did not forecast the entire curve at once; rather, it predicted only *one* next measurement based on knowledge of the preceding 50 actual measurements.

Lets do it more fair. Suppose we have a starting point for predictions and had no access to real measurements beyond that point. Utilizing the previous 50 measurements, we make *one* initial prediction.

Figure 54: Sinusoidal time series with added noise.

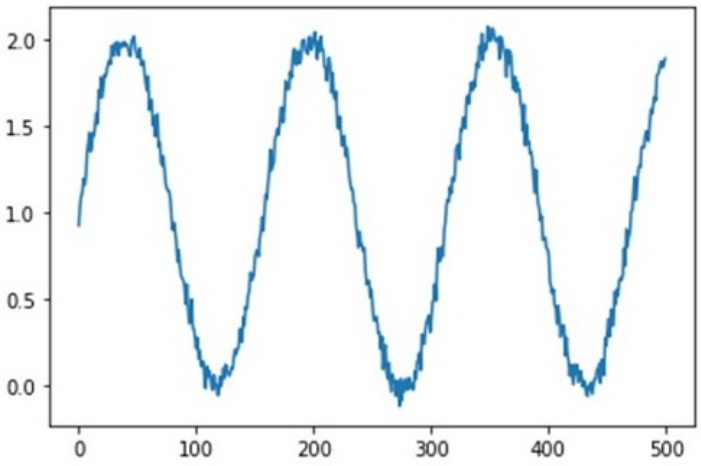


Figure 55: Prediction of the network vs actual data.

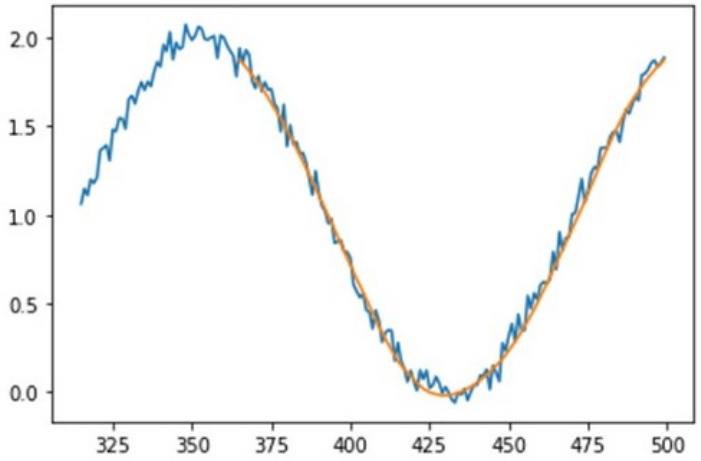
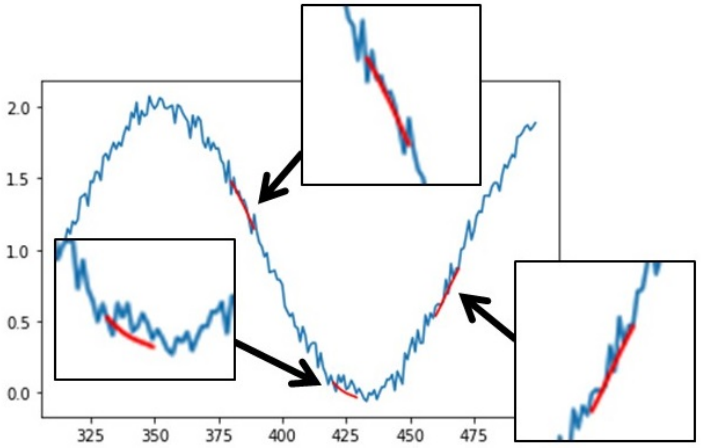


Figure 56: 10-steps prediction starting from an arbitrary chosen point.



Subsequently, we treat this prediction as a fictitious measurement, appending it to the 49 previous actual measurements (totally 50 required) to make the subsequent guess. This process continues iteratively for the

requested number of steps.

Of course, such predictions based on predictions will fail sooner or later but looking into the future for, say 10 steps, is well possible.

Why Bond was a bit skeptic about the predictions? Maybe he meant forecasting something more complicated than a sin function?

7.3 Used codes

Listing 17: Time series prediction with LSTM networks.

```
import numpy as np
import matplotlib.pyplot as plt

# Example data
Length =500
aXis =np.arange(Length)
rng = np.random.default_rng()
data =np.sin(aXis*20/Length) + 1 +rng.normal(0, 0.05, aXis.shape[0])
plt.plot(aXis,data)
plt.show()
data =data.reshape(-1,1)
print(data.shape)

# Prepare data batches
Interval =50
X,y = [],[]
for i in range(Interval, len(data)):
    X.append(data[i-Interval:i, 0])
    y.append(data[i, 0])
print('number of available series',len(X))

# Split to training and test sets
train_size = int(len(X) * 0.7)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
print('train series',len(X_train),'test series',len(X_test))

X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
print(X_train.shape, y_train.shape)

# Build model
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Flatten
model = Sequential()
# Adding LSTM layers
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=True))
# Adding Dropout to supress overfitting
model.add(Dropout(0.2))
# Adding a Flatten layer before the final Dense layer
model.add(Flatten())
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

# Train the model MIGHT TAKE TIME !
history = model.fit(X_train, y_train, epochs=10, batch_size=25, validation_split=0.2)
```

```
# Visualize training epochs
History = history.history
plt.plot(History['loss'], label='Train_loss')
plt.plot(History['val_loss'], label='Val_loss')
plt.xlabel('Epoch')
plt.title('Loss')
plt.show()

# Predict test data (One step prediction)
X_test, y_test = np.array(X_test), np.array(y_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_pred = model.predict(X_test)
print(X_test.shape,y_pred.shape)

plt.plot(aXis[train_size:],data[train_size:])
plt.plot(aXis[train_size+Interval:],y_pred.flatten())
plt.show()

# Multi step prediction
def multipass_prediction(X_test,rel_Point,Steps,Interval):
    current_batch = X_test[rel_Point,:,:]
    current_batch =current_batch.reshape(1,Interval,1)
    predictions =np.zeros(Steps)
    for i in range(Steps):
        one_prediction = model.predict(current_batch)
        print(one_prediction)
        one_prediction =one_prediction.reshape(1,1,1)
        predictions[i] =one_prediction[0][0]
        current_batch = np.append(current_batch[:, 1:, :], one_prediction, axis=1)
    return predictions

def show_multipass(abs_Point,X_test,train_size,Interval,Steps):
    y_pred = multipass_prediction(X_test,abs_Point-train_size-Interval,Steps,Interval)
    plt.plot(aXis[abs_Point:abs_Point+Steps],y_pred,color='red')

plt.plot(aXis[train_size:],data[train_size:])
Steps =10

# Predict for 10 steps starting given points
show_multipass(380,X_test,train_size,Interval,Steps)
show_multipass(420,X_test,train_size,Interval,Steps)
show_multipass(460,X_test,train_size,Interval,Steps)

plt.show()
```

8 To find a needle in a haystack

8.1 James Bond in troubles

Upon entering the room, I was startled to find James Bond on all fours, frantically crawling on the floor. Fear gripped me as I wondered if the building was besieged by enemies, poised to unleash a barrage of gunfire.

Figure 57: James Bond in troubles.

"What's wrong? Are we under attack?" I stammered, my voice barely audible in the tension of the moment.

"Quiet!" Bond commanded sharply. "It's worse than you think. Don't move. Stay right where you are."

"What's happened?" I whispered, my shock intensifying.

"A screw from my glasses has fallen to the floor. Don't step on it!" Bond explained tersely as he slowly rose from the ground. With practiced ease, he produced a miniature camera from his pocket, capturing an image and manipulating the device. "Aha! There it is," he declared triumphantly, plucking the once-invisible screw from the floor, his expression smoothing into satisfaction.

"How did you manage to locate such a small object?" I marveled.

"I have a high-resolution camera equipped with an embedded neural network capable of swiftly identifying any requested object within its field of view," Bond revealed.

"Remarkable technology! I can envision its applications in locating hidden aircraft or missiles in space photographs," I said, impressed by the possibilities.

"Indeed, it can" Bond agreed, "though its primary function is to locate my lost glasses screws on the floor."

8.2 Convolutional neural networks catch objects

Intrigued by this technological marvel, I endeavoured to replicate it on my laptop.

I simulated images of a screw in various locations and orientations. Then, I mimicking the dirty floor in Bond's room (it was quite cluttered, by the way).

Employing a simple convolutional neural network, I devised a method to scan the floor sector by sector, successfully localizing the screw while disregarding unrelated objects.

Works fine! It is probably as good a network as that embedded in the Bond's camera.

Figure 58: The secret agent has in his arsenal more tricks than you can expect.

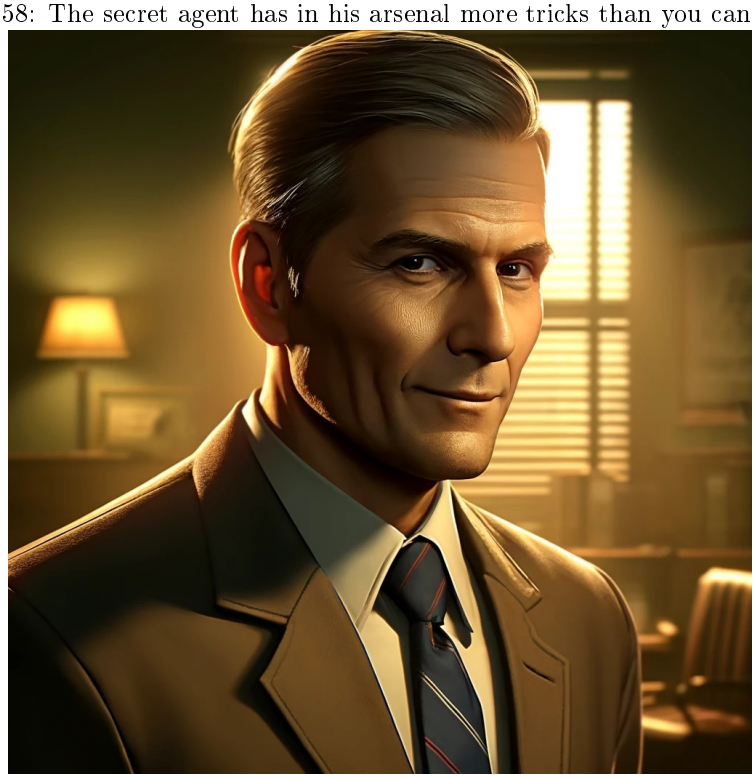


Figure 59: The cluttered floor consisting of 16 x 16 cells each with a piece of dirt.

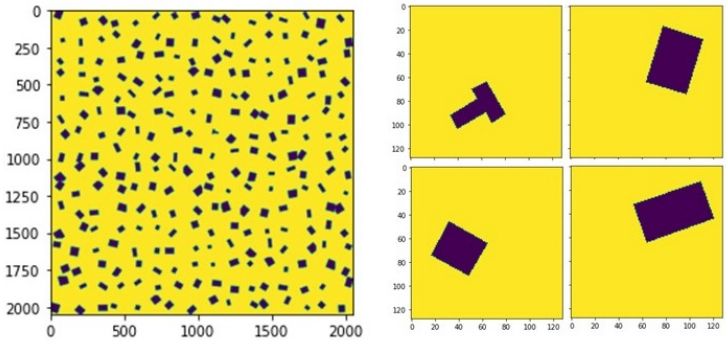
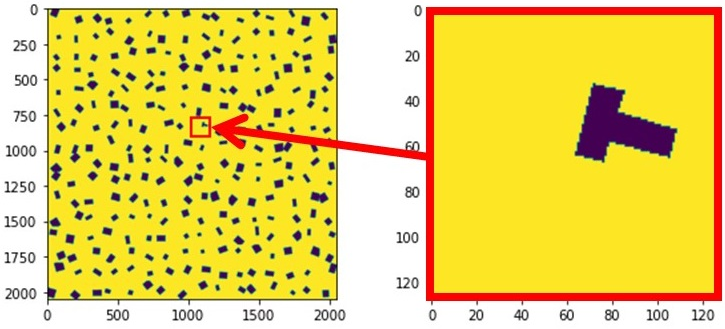


Figure 60: A screw is successfully localized by the convolutional neural network.



It's been trained to find lost screws. Perhaps we should redirect its capabilities toward more valuable pursuits? Searching for lost friends? Good moods? Happiness?

8.3 Used codes

Listing 18: Simulation of chaotic objects spread on the floor.

```
import numpy as np
import matplotlib.pyplot as plt
import random as r
from PIL import Image, ImageDraw

def rotated_fig(fig,x,y,angle):
    theta = (np.pi / 180.0) * angle
    R = np.array([[np.cos(theta), -np.sin(theta)],
                  [np.sin(theta), np.cos(theta)]])
    offset = np.array([x, y])
    transformed_fig = np.dot(fig, R) + offset
    return transformed_fig

def rect(x, y, w, h, angle):
    rect = np.array([(0, 0), (w, 0), (w, h), (0, h), (0, 0)])
    return rotated_fig(rect,x,y,angle)

def screw(x, y, w, h, d, l, angle):
    w1 = (w-d)/2
    w2 = w - w1
    rect = np.array([(0,0), (w,0), (w,h), (w2,h), (w2,l), (w1,l), (w1,h), (0,h), (0, 0)])
    return rotated_fig(rect,x,y,angle)

def random_rect(size):
    l_min = size/6
    l_max = size/2
    margin =size/2

    w = l_min + np.random.random()*(l_max -l_min)
    h = l_min + np.random.random()*(l_max -l_min)

    x = margin + np.random.random()*(size - 2*margin)
    y = margin + np.random.random()*(size - 2*margin)
    angle = np.random.random()*360

    return rect(x, y, w, h, angle)

def random_screw(size):
    margin =size/2

    x = margin + np.random.random()*(size - 2*margin)
    y = margin + np.random.random()*(size - 2*margin)
    angle = np.random.random()*360

    return screw(x,y,size/4, size/10, size/10, size/16*5, angle)

def draw_cell(size,screw=False):
    # numpy 2D array
    data =np.ones((size,size))

    # convert the numpy array to an Image object.
    img = Image.fromarray(data)

    # draw a rotated rectangle or screw on the image.
    drawing = ImageDraw.Draw(img)
    if screw ==True:
        fig = random_screw(size)
    else:
```

```
        fig = random_rect(size)
    drawing.polygon([tuple(p) for p in fig], fill=0)

    #convert back to np array
    return np.asarray(img)

def draw_floor(floor_size, cell_size):
    floor =np.ones((cell_size*floor_size,cell_size*floor_size))

    screw_x = r.randint(0,floor_size)
    screw_y = r.randint(0,floor_size)

    for x in range(floor_size):
        for y in range(floor_size):
            if x == screw_x and y == screw_y: there =True
            else: there =False
            cell = draw_cell(cell_size,screw =there)
            floor[y*cell_size : (y+1)*cell_size,x*cell_size : (x+1)*cell_size] = cell
            if there ==True:
                plt.imshow(cell)
                plt.show()
    return floor, screw_x, screw_y

if __name__ == "__main__":
    floor_size =16
    cell_size =128

    cell = draw_cell(cell_size)
    plt.imshow(cell)
    plt.show()

    floor,screw_x, screw_y = draw_floor(floor_size, cell_size)
    plt.imshow(floor)
    plt.show()
    print('screw at x =',screw_x,' y =', screw_y)
```

Listing 19: Convolutional neuronal network localizing a screw on the floor.

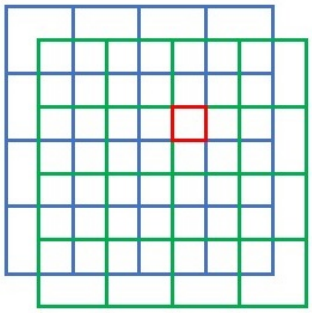
```
from Needle_simulate import * # the previous script simulating objects on the floor
                              # must be imported here
from tensorflow.keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from keras.optimizers import Adam
from matplotlib.patches import Rectangle

floor_size =16
cell_size =128

floor, screw_x, screw_y = draw_floor(floor_size, cell_size)
plt.imshow(floor)
plt.show()
print('screw at x =',screw_x,' y =', screw_y)

def rebin_2D(arr,Bin):
    Height,Width =arr.shape
    shape = (Height//Bin, Bin, Width//Bin, Bin)
    # for Bin=2: H/2     2       W/2      2
    return arr.reshape(shape).mean(3).mean(1)

def build_database(Capacity,cell_size,Bin):
    data =np.zeros((Capacity,cell_size//Bin,cell_size//Bin))
    labels =np.zeros((Capacity,1),dtype=bool)
```

```
    for i in range(Capacity):
        there = r.choice([True, False])
        cell = draw_cell(cell_size,screw =there)
        cell = rebin_2D(cell,Bin)
        data[i,:,:] =cell
        labels[i,:] =there
    data.shape = data.shape +(1,)
    return data, labels

Bin =4
data, labels = build_database(1000,cell_size,Bin)
print(data.shape,labels.shape)

# simplest convolutional network
model = Sequential([
      Conv2D(16, (3,3), activation='relu', input_shape=(cell_size//Bin, cell_size//Bin,1),
          padding='same'),
      MaxPooling2D(2,2),
      Conv2D(32, (3,3), activation='relu', padding='same'),
      MaxPooling2D(2,2),
      Conv2D(64, (3,3), activation='relu', padding='same'),
      MaxPooling2D(2,2),
      Flatten(),
      Dense(256, activation='relu'),
      Dense(1, activation='sigmoid')
                ])
#model.summary()

model.compile(loss='binary_crossentropy',
              optimizer=Adam(learning_rate=0.0005), metrics='accuracy')

history = model.fit(data, labels,
                    epochs=20,
                    )

plt.plot(history.history['loss'], label='Train_loss')
plt.show()

def fragm(floor,x,y,cell_size):
    return floor[y*cell_size : (y+1)*cell_size,x*cell_size : (x+1)*cell_size]

def check_floor(floor,floor_size, cell_size, Bin):
    data =np.zeros((floor_size**2,cell_size//Bin,cell_size//Bin,1))

    for x in range(floor_size):
        for y in range(floor_size):
            cell = fragm(floor,x,y,cell_size)
            cell = rebin_2D(cell,Bin)
            data[y + x*floor_size,:,:,0] =cell

    labels = model.predict(data)
    labels = (labels >0.5)
    found_index = np.argmax(labels)
    found_x = found_index //floor_size
    found_y = found_index - found_x*floor_size

    return found_x, found_y

found_x, found_y = check_floor(floor,floor_size, cell_size,Bin)
print('found at x =',found_x,'y =', found_y)

x0 = found_x * cell_size
```

```
y0 = found_y * cell_size
plt.imshow(floor)
rect = Rectangle((x0,y0),cell_size,cell_size,linewidth=2,edgecolor='r',facecolor='none')
plt.gca().add_patch(rect)
plt.show()

cell = fragm(floor,found_x, found_y,cell_size)
plt.imshow(cell)
plt.show()
```

Ziming:
Hi Pavel, I checked your code and found some tricking there. You break the image on the fixed fragments. If the screw comes near the boundary of the fragment it is not found.

Figure 61: Scanning over two grids. Objects near the border of one grid appear withing the depth of another one. Objects can be localized with the double precision if they are in depth for both grids or with the standard precision if they are near the borders.



Pavel:
This issue is easily fixed by adding the second grid shifted relative the first one. Such network will catch 95% cases and eventually localize the object more precisely.

This is however not essential. You can infinitely improve the precision and robustness of the neural network, especially if you are paid for that. My manuscript is not a tutorial on machine learning but rather a key to understanding of what is going on. I see from your question that you already got the point.

9 James Bond and the Struggle with Common Sense

9.1 Bond lost his way

Figure 62: Reading is fun.

Countless times, James Bond had guided my thoughts and taught me lessons I would never forget. However, I take pride in the fact that, on one occasion, I managed to help him out of a rather intricate situation.

I happened upon James standing in the middle of a busy road, completely oblivious to the torrent of cars swerving dangerously around him. "What are you doing there?" I called out, alarmed.

"Hello," he replied absently, "I'm reading a book by the Greek philosopher Zeno."

"Please, get out of the traffic! Take care of yourself!"

"That's impossible," he said calmly, his gaze fixed on the book.

"Listen! To traverse a path, you must first cover half of it. To do that, you must cover a quarter of the way, and then an eighth, and so on, infinitely. A person cannot perform an infinite number of actions, and thus, movement itself is impossible."

"Let's discuss this after you've moved from that dangerous spot!" I urged.

"But I can't find any mathematical contradictions in Zeno's argument."

At that point, I lost my patience and shouted more forcefully than I intended, "Are you crazy? Get out of there!"

"It seems you're asking for the impossible, Sir," James replied with a calm dignity.

I quickly changed my approach and appealed to his common sense. "I can confidently assure you that movement is possible. I've experienced it a number of times, and you can too. Just take a step!"

With evident caution, he moved forward slowly, carefully stepping off the road.

Figure 63: James Bond in ancient Greece.

Figure 64: Back to reality.

"I'm used to trusting mathematical predictions rather than feelings or intuition," James said, a hint of

embarrassment in his voice.

"I think we still need to balance abstract concepts with common sense," I said, giving him a reassuring pat on the shoulder.

"Right," he acknowledged thoughtfully.

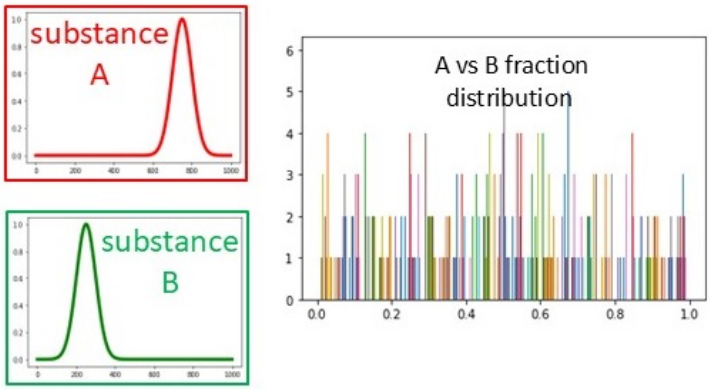
"By the way, the PCA method we discussed so intensively often yields quite counterintuitive results that must be significantly reinterpreted with a dose of common sense. . . "

9.2 From Abstraction to Common Sense

We continued our conversation at my place, where James began to speculate.

"Suppose we have a mixture of substance A and substance B, each with its own unique spectral signature. To keep things simple, let's assume A and B are distributed randomly in equal proportions. As usual, we apply PCA—in this case, the uncentered version—and reduce the dimensionality of the dataset down to two. That is, we extract two spectral components that can describe any data distribution in this mixture."

Figure 65: Substances A and B with their unique spectral signature.



"Since there are only two variables, we can easily plot the data distribution over these two components. The data variation would form a straight line, as the content changes linearly from A to B. But what does this 2D plot mean?" James quickly sketched the results of the PCA.

"This is a space where each point represents a spectral signature, and all of these signatures are linear combinations of just two 'basic' spectra—the signatures of the two principal components."

"Now, look more closely at this 2D set of spectral signatures. Most of them are physically impossible, as a spectral feature cannot be negative. Even the spectral signature of the second principal component falls into the forbidden domain."

"This is a catastrophe! How can we use PCA after that?" I exclaimed.

"Steady! We still can," James replied. "We can always find a linear combination of principal components that adheres to all physical constraints. In this particular case, we can simply look at the data distribution in the 2D plot and manually select two points at the edges of the data spread. Let's call them endmembers. Since all points in our 2D space are linear combinations of one another, we can express them in terms of these two reference points, instead of the abstract principal components. These new basis points would correspond very closely to the ground truth we set in the simulations."

"Great! But why do we need principal components if we end up recasting everything into other qualities anyway?"

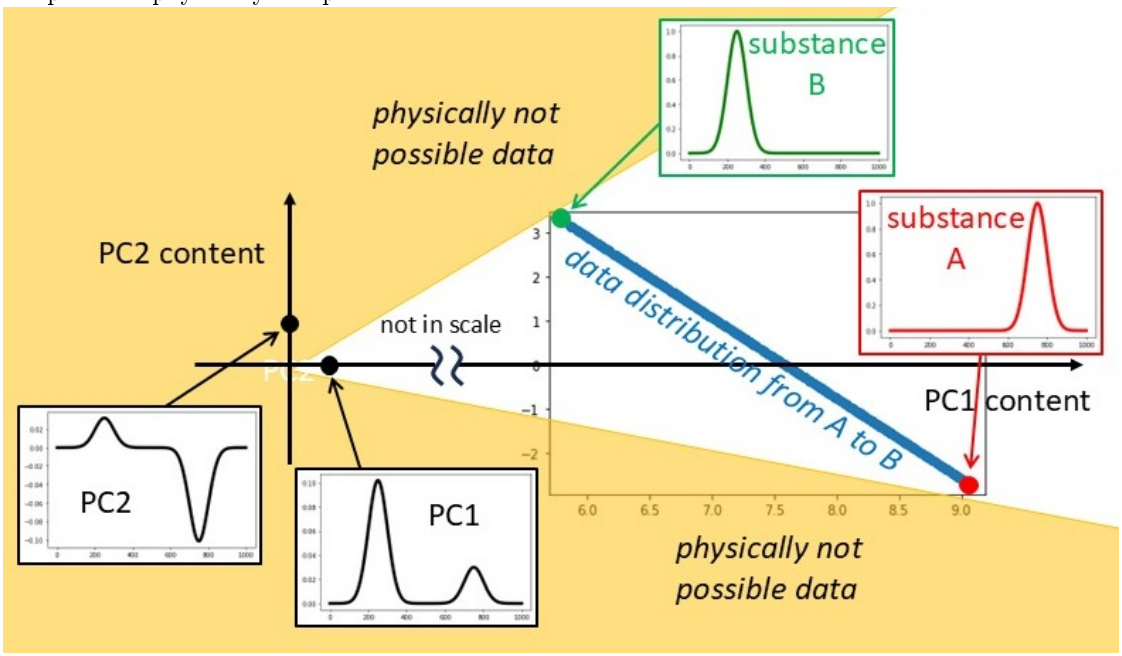
"Ah... that's more of a technical issue. But we can take another approach. Let's start by searching for two reference points that 1) satisfy physical constraints, such as non-negativity, and 2) ensure non-negative fractions in all available data points. This method is called Non-negative Matrix Factorization (NMF)."

Bond typed something into the computer, and after a brief moment, a satisfied smile crossed his face.

"The results are quite similar to what I obtained with PCA followed by endmembering. However, the algorithms behind NMF are more complex and less robust than PCA. So, using good old PCA might not be such a bad idea after all. And, I should warn you that the NMF solution might be not unique."

"Doesn't matter!" I said enthusiastically. "From now on, I will adopt this wonderful approach: search only for solutions that are non-negative."

Figure 66: Scatter plot of two principal components and the actual data distribution. Most of spectra in this 2D space are physically not possible.



9.3 Back to Abstraction

"Wait a moment!" James said thoughtfully. After a brief pause, he added, "I think the restriction of strictly non-negative fractions in each data point could, at least partially, be relaxed."

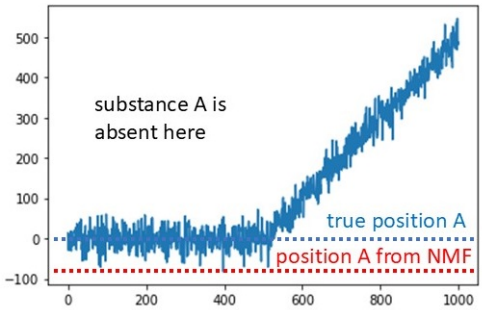
"Negative content?" I asked, trying to be as gentle as possible. "James, are you starting to feel unwell again, like when you were crossing that street?"

"Not at all," Bond smiled. "And I'm going to show you that this idea is consistent with both common sense and mathematics."

"Imagine that substance A is absent over a large portion of the dataset. The content there must be zero, and indeed, NMF would produce zero. However, if the data is corrupted by significant noise, the output can't be exactly zero. Of course, PCA, NMF, and other dimensionality reduction methods reduce noise, but they never eliminate it completely. In the region we're discussing, the content shouldn't be zero but should instead fluctuate randomly around zero."

"If we insist on strict non-negativity of the content, the position of reference point A must be shifted from the ground truth. The bias increases with the noise level. Even more so, for the same levels of noise, the bias would vary randomly depending on the largest outlier produced by the noise."

Figure 67: Non-negative Matrix Factorization might create a bias in the situation of high noise.



"Are there methods to account for noise in this situation?"

"There are," James replied. "I'll need to head back to headquarters to check it. You know, my colleagues are unmatched experts in creating noise and thriving in noisy environments..."

9.4 Used codes

Listing 20: Non-Negative Matrix Factorization.

```
import numpy as np
import matplotlib.pyplot as plt
#from scipy.stats import kurtosis
from sklearn.decomposition import TruncatedSVD, NMF

def gaussian(x, mu, sig):
    return np.exp(-np.power((x - mu)/sig, 2.) / 2)

def signal(D,Start,End,Fract,NoiseSigma):
    Sigmas =5 #+-sigma in range (Start:End)
    Sig =(End -Start)/2/Sigmas
    Mu = (Start +End)/2

    spec =np.arange(D)
    spec = gaussian(spec,Mu,Sig)*Fract

    if NoiseSigma >0:
        spec += np.random.normal(0,NoiseSigma,D)

    return spec

Size=32
Depth =1000
Sigma =0.0

plt.plot(np.arange(Depth),signal(Depth,0,Depth/2,1,0),linewidth=5,color ='green')
plt.show()
plt.plot(np.arange(Depth),signal(Depth,Depth/2,Depth,1,0),linewidth=5, color ='red')
plt.show()

distribution =np.random.uniform(0,1,size =(Size,Size))
plt.hist(distribution,bins=25)#,color ='skyblue')#'auto')
plt.show()


def make_SI_2feature(im,Depth,Sigma):
    Height,Width =im.shape
    imSI =np.zeros((Height,Width,Depth))

    for y in range(Height):
        for x in range(Width):
            #print(x,y)

            Frac1 = (1+im[y,x])/2
            Frac2 =1-Frac1
            spec = signal(Depth,0,Depth/2,Frac1,Sigma) #add 1st feature
            spec += signal(Depth,Depth/2,Depth,Frac2,Sigma) #add 2nd
            imSI[y,x,:] =spec
    return imSI


SI = make_SI_2feature(distribution,Depth,Sigma)
SI.shape =(Size*Size,Depth)
Components =2
model =TruncatedSVD(n_components=Components)
model.fit(SI)
scores = model.transform(SI)
loadings =model.components_
```

```
print(scores.shape,loadings.shape)

plt.plot(np.arange(Depth),model.components_[0,:],linewidth=5,color='black')
plt.show()
plt.plot(np.arange(Depth),-model.components_[1,:],linewidth=5,color='black')
plt.show()

plt.scatter(scores[:,0],scores[:,1])
plt.show()

print(model.components_.shape, model.singular_values_)
plt.plot(np.arange(Depth),model.components_[0,:])
plt.plot(np.arange(Depth),model.components_[1,:])
print(np.sum(model.components_[0,:]**2))
plt.show()

model2 =NMF(n_components=Components)
scores = model2.fit(SI)
scores =model2.fit_transform(SI)
print(scores.shape)
print(np.min(scores[:,0]),np.max(scores[:,0]))
print(np.min(scores[:,1]),np.max(scores[:,1]))
plt.plot(np.arange(Depth),model2.components_[0,:])
plt.plot(np.arange(Depth),model2.components_[1,:])
plt.show()
```

Balagusla:

The comment of 'James Bond' about noise which captures the negative area is strange. The noise is Poissonic and nonnegative even at very low signal.

Pavel:

Indeed, noise in experimental spectra is Poissonian. But most of methods in multivariate statistical treatment imply the Gaussian noise that requires some data pre-processing, typically rescaling. Therefore, the treated spectra might shows slight negative counts. More important: we were talking about contents, not about spectra. The extracted content of a substance is mostly distributed as Gaussian, thus may be negative.

Anonymous:

This story implies a mental disorder. I have another image of James Bond.

Pavel:

The last thing I want is to tarnish the image of James Bond. I don't believe that a small flaw can tarnish the reputation of a great man. In fact, I've observed that many brilliant individuals often display, quite honestly, some eccentricities. Whether this is an inherent trait of genius, I'll leave for you to judge...

Steve:

I dont agree. NMF is non-negative factorization -> obeys Poissonian stat, it cannot go to negative. I think the last figure is incorrect.

Pavel:

Lets see where is the root of confision. Unfortunately, people might understand *very different techniques* under 'NMF'. Look at the code, I use sk-learn nmf(...) that, in fact, may employ *very different algorithms* depending on the options. I (as 95 percent of others) used the default beta-loss='frobenius', which is essentially minimizing the squared differences between observation and NMF results. That is relatively simple algorithm that just penalizes the negative solutions. 'Frobenius' unavoidably implies Gaussian noise in solution and (as James mentioned) can create a bias. However, if you set beta-loss='kullback-leibler', this activates completely different algorithm that explicitly accounts for Poissonian noise. The solution would be perfectly correct and unbiased. But don't be surprised if it computes 100 times longer...

10 James Pisses Me Off

10.1 Back to Childhood

Figure 68: Back to childhood.

One evening, I knocked on James's door. After a distracted "Please..." I entered, only to find him in an unusual situation. James was sitting on the floor, completely absorbed in building a model of an old sailing ship. He looked so much like a child at play that I couldn't help but laugh.

"Back to childhood, are we?" I teased.

James seemed a bit piqued.

"One might think you're not addicted to modelling..." he retorted.

"I'm not," I replied. "I deal with real things, I don't waste time on games and models."

"What do you call real things, my friend?" James asked with a twinkle in his eye.

"Same things you call real — this room, this desk, you, standing by the desk..." I trailed off, but he cut me off.

"You never see me," he interrupted sharply.

"What do you mean? I see you right now, and—"

"What you see," he continued softly, "is just a distribution of light intensity — a rather weird distribution — that your brain associates with James Bond based on your previous experiences."

"No, I don't make any associations," I objected. "I'm just seeing you directly."

Figure 69: In dark room you might discover almost anything.


"No. You have a model of 'James Bond' in your head," Bond insisted. "This model may or may not have anything in common with the real object."

"James, your fantasies are sometimes interesting, but..." I trailed off.

Bond, looking tired, like someone explaining the obvious to others, lost his patience. "And now, do you see me?" he asked, suddenly switching off the light.

"Of course, some details are harder to discern, but I can still clearly see you in general." I didn't give up, adding more and more arguments, but faced only icy silence.

Unexpectedly, I felt a hand on my back. It was James, who had cleverly moved to another corner of the room. When the light came back on, I found I'd been arguing in front of a hat hanging on a vase atop an old secrétaire.

"So, you think you're not modelling reality based on a few weird signals coming from the external world?" James continued.

"You fooled me with this old spy trick, but it doesn't prove anything. Yes, modelling is sometimes useful, but ..."

"Sometimes? More like always. You look like a scholar about to publish an article titled 'Model-based Quantification of the Baldness Process.' In fact, all scientific knowledge is model-based." After a brief pause, he added, "And not just scientific knowledge — any knowledge, skills, beliefs, and so-called common sense are nothing but models."

10.2 Model Might Miss Essential Things

Honestly, James pissed me off. Even when I got home, I couldn't calm down. What nonsense he sometimes spouts! Does he really issue absurd, harmful, and even extremist ideas just to appear wise, original, and mysterious? And all with cheap tricks!

To calm down, I decided to do what I love — data analysis. Let's say we measure light from a distant supernova explosion. Theorists predict that its time dependence should follow a Gaussian curve. All we need to know about this event are the parameters of the Gaussian — the width and the magnitude — and once we have those, we know everything about the supernova. I quickly wrote a code, and voilà, the fit was perfect! I was especially proud of my model because it outputted the variance of the parameters - magnitude a, explosion peak moment μ, and width σ. This serves as a reliability measure; if the variances exceed a certain threshold, we reject the model.

But then I tested the robustness of the model by introducing additional intensity dips just before and after the explosion's peak. I expected that the estimated reliability would degrade, prompting us to reject the model. Unfortunately, the results were mixed. Some parameters showed worse reliability, others improved,

Figure 70: Supernova explosion fits perfectly theory.

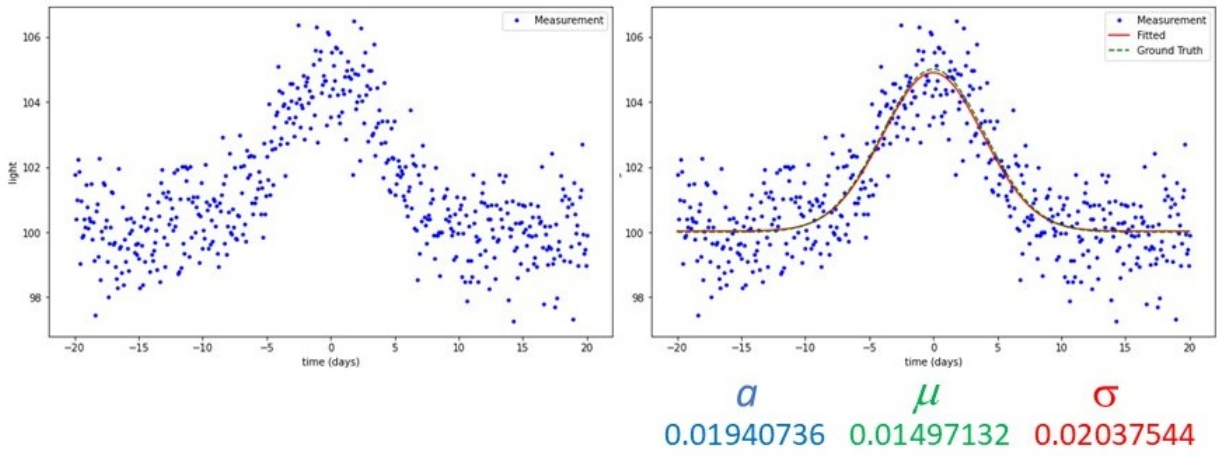
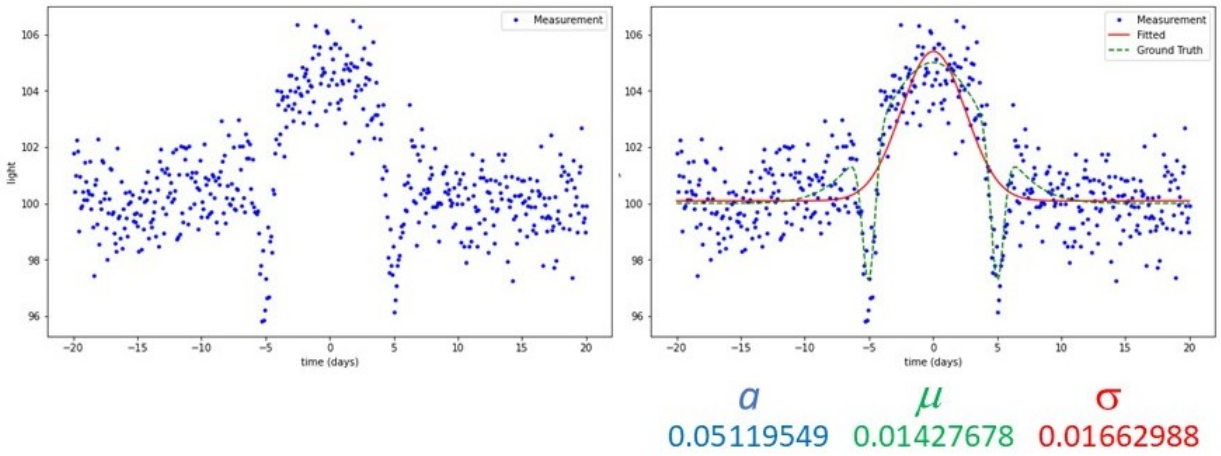


Figure 71: Another supernova shows extra features that are not captured by the model.



like the model pretended to handle the data better. What went wrong? I clearly saw the extra features, but the model didn't. It simply filled the non-fitting gaps and remained content with the result.

So maybe James was partially right — models can sometimes miss essential things.

But generally, he's wrong. I'm not a model. I'm a data scientist, an experienced professional. I'm, last but not least, a member of the Royal Scientific Society. I see things as they are — I don't just fill gaps in reality. Or do I?

10.3 Used codes

Listing 21: Fitting a supernova flash to the model.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


def gaussian(x, a, mu, sigma, c):
    return a * np.exp(-((x - mu) ** 2) / (2 * sigma ** 2)) + c


# Generate synthetic noisy Gaussian data
```

```
time = np.linspace(-20, 20, 500)
true_params = [5, 0, 4, 0] # [amplitude, mean, stddev, offset]
I_true = gaussian(time, *true_params)

# Add extra features
I_true += gaussian(time,-5,-5,.5,0)
I_true += gaussian(time,-5, 5,.5,0)

# Add noise and background
np.random.seed(0)
noise = np.random.normal(0, 1, time.size)
background =100
I_noisy = background + I_true + noise

# Fit the noisy data
initial_guess = [4, 0, 1, 0] # Rough initial guess for parameters
fitted_params, pcov = curve_fit(gaussian, time, I_noisy, p0=initial_guess)
I_fitted = gaussian(time, *fitted_params)
print(np.diag(pcov))

# Plot the data and fit
plt.figure(figsize=(10, 6))
plt.plot(time, I_noisy, 'b.', label='Measurement')
plt.plot(time, I_fitted, 'r-', label='Fitted')
plt.plot(time, background + I_true, 'g--', label='Ground Truth')
plt.legend()
plt.xlabel('time (days)')
plt.ylabel('light')
plt.show()
```

Anonimous:

Not you neither your 'James' see the truth. Both are idiots. Do Buddhistic meditation!

Pavel:

I wanted first to trash this comment but finally approved it because of its remarkable, amazing style.

Jeff:

The right criterion to evaluate a model is a squared sum of fit. Isn't it?

Pavel:

That is not as essential. There are many metrics. The squared sum is 496.3 for the 'good' fit and 884.9 for the 'filling gaps' model. Thus, we still cannot surely set the threshold between an adequate and a something-missing model.

Neville:

Pavel, are you a member of a Royal Society like Isaak Newton?

Pavel:

I afraid, there is a confusion. In a Royal Society of London formerly headed by Isaak Newton there is no word 'Scientific'. I mean the Royal Scientific Society founded in 1973 by the king Mohamed VII in Equatorial Guinea. I have recently received a mail from them electing me as a member in recognition of my scientific achievements. I immediately sent them 20 dollars as a membership fee and now have every right to call myself a member of this glorious society.

11 Gaps in Minds

11.1 Shocking Experiment

I had been thinking a lot about the topic James and I had strongly disagreed on last time. Although I hadn't changed my mind, James's position had become clearer to me.

When we met again, I admitted that models can, at times, unnoticedly fill in gaps in our perception of the surrounding world.

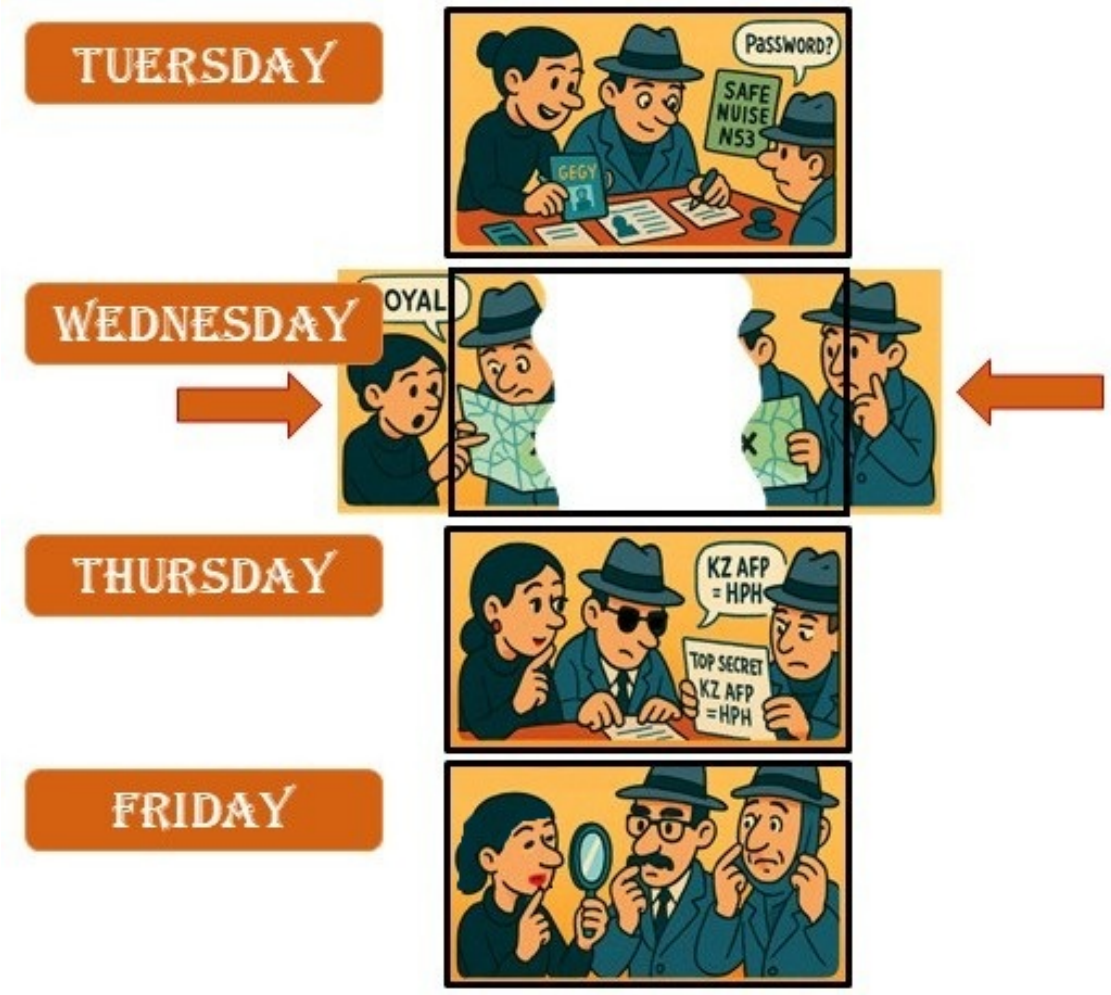
"But it doesn't happen that often," I added.

"Quite often," he objected. "They fill gaps not only in space, but also in time."

"In the past, our agency conducted an experiment," he continued. "You know, we regularly organize training sessions for agents. They work in groups — typically three people — for several weeks, each day of the week devoted to a specific activity. For example, on Tuesdays they're trained to memorize secret addresses and passwords; on Wednesdays, they practice rapid orientation using geographical maps; on Thursdays, they encode and decode confidential reports, and so on."

"After a few weeks of training, all three agents were given a harmless drug that caused them to sleep for about 30 hours, effectively skipping the entire Wednesday. They woke up Thursday morning, quickly realizing something was off. At first, they exhibited signs of anxiety, but soon they forced themselves to carry on as if everything was normal."

Figure 72: Concentrated on everyday duties the spies did not notice that somebody removed the entire Wednesday from their life.



"Did they try to share their suspicions with each other?" I asked.

"No," James said. "Each of them pretended nothing had happened. They hid their unease and quickly became fully immersed in the regular Thursday activities. It was as if they collectively chose to ignore what had occurred.

And here's the most interesting part: after the training was over, they each wrote detailed reports describing their activities — day by day — including a day that never actually existed. I must stress, these were highly trained and scrupulously honest individuals."

"Amazing!" I said. "By the way, I didn't know your agency was involved in basic psychological research."

"It is," he nodded. "And for very practical reasons. Since that experiment, the agency no longer trusts a posteriori reports. Agents are now required to make brief, real-time (encoded) notes throughout a mission — accurately recording the current time as they go."

11.2 Exploitative vs Explorative Strategies

The most peculiar detail in the story told by James was the fact that none of the participants in the experiment tried to share their doubts with others. Were they afraid of appearing completely insane? Maybe not. Perhaps they were somehow hypnotized by what is often called common opinion, even when it contradicted their own subjective experience.

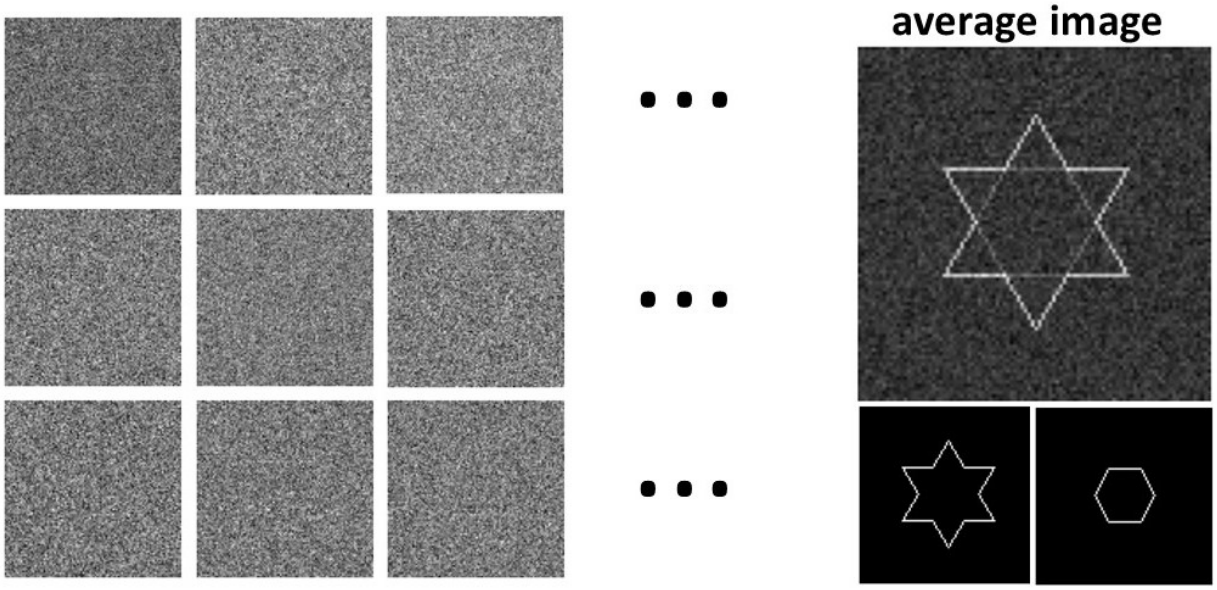
Indeed, it is nearly impossible to question a well-established, widely accepted model dictating that Tuesday is followed by Wednesday. This model is very useful in 99.99 percent of cases. However, if you once clearly notice that Thursday comes after Tuesday, maybe it's better to admit that fact honestly and explore what consequences might arise.

This suggests that your strategy should not be 100 percent exploitative (i.e., relying solely on established concepts), but partially explorative — actively checking whether factors not previously included in the model, or even completely unknown ones, might better explain the observations.

Let me sketch an example from structural biology. In that field, scientists collect thousands of TEM images of proteins. Each individual image has such weak contrast that it's impossible to discern any structure. But by correlating many images taken from different orientations and averaging them, the underlying structure emerges quite clearly.

Let's try to mimic this technique. Suppose we've got a number of TEM images that look like this:

Figure 73: Each individual bio TEM image shows almost nothing, but averaging all available data shows a pattern.



Not very impressive, right? But if we average all these noisy images — omitting, for simplicity, the complex steps of projection, orientation, and alignment — some pattern begins to appear, resembling known reference structures that we call here "stars" and "hexagons".

We might even guess that these structures are present in a 2:1 ratio.

So far, so good! Let's fit each individual image to either the star or hexagon reference and count how many of each we have. Then, we sum up the resulting classes and compare the averages with the reference structures.

The averaged pattern for the star looks almost identical to the reference — though a careful examination reveals a few excessive features. Maybe our classification algorithm wasn't perfect. Further - worse... The average pattern for the hexagon is so different from the reference that it's hard to ignore.

At this point, it's probably time to switch from an exploitative to an explorative strategy. Suppose, that there are no hexagons at all. It's not a strongly grounded suggestion, but let's see where it leads. If that's the case, then everything that doesn't match the star pattern must belong to something else — a previously unidentified class.

After some hard work, we arrive at another picture: a mixture of stars and something new — triangles!

Figure 74: Matching to well-known reference structures does not work as well...

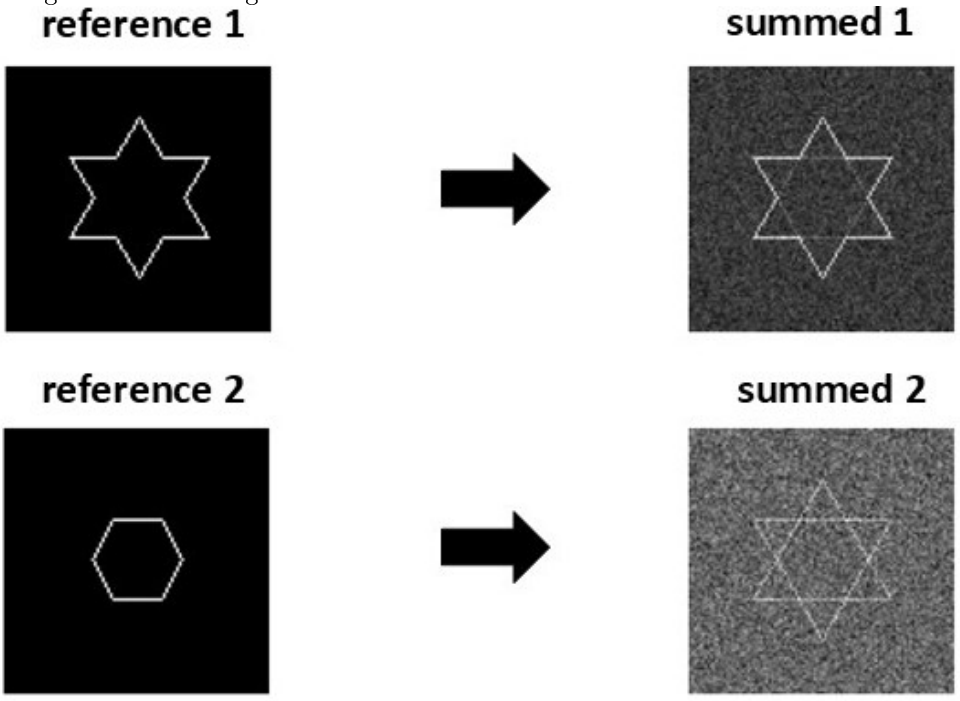
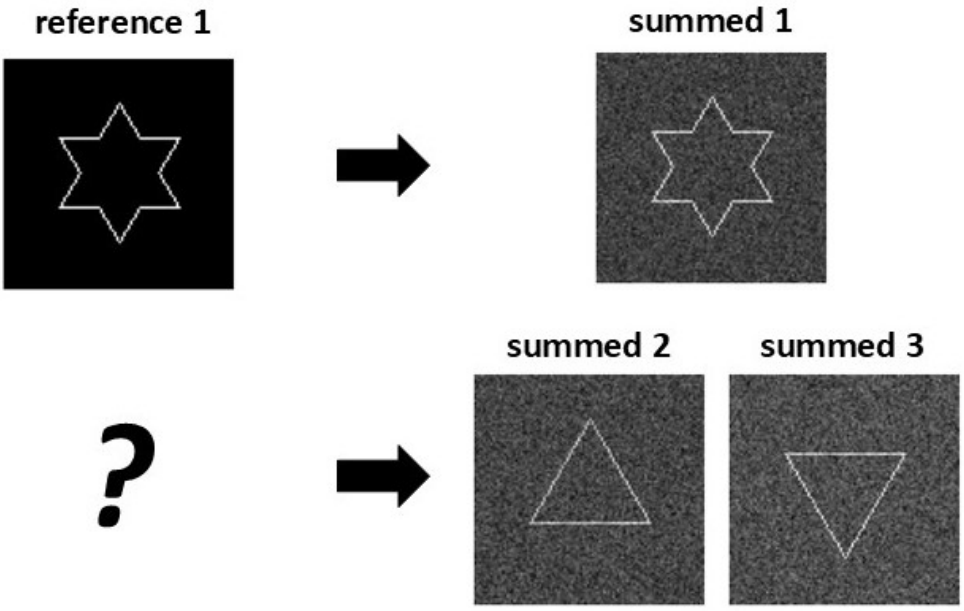


Figure 75: Exploring new patterns makes a discovery.



Now the observations are coherent. The star pattern is well reproduced and the new triangle patterns explain all the rest. Since the triangles appear in two mirrored variants, we conclude there is a 1:1 mixture of stars and triangles (See the script for details on how this quantification is performed).

Here's the funny part: the new interpretation produces exactly the same average image as our initial conservative hypothesis.

Conclusion: We took the risk of exploring an unknown domain — and it paid off. A new protein structure is discovered!

11.3 Used code

Listing 22: Mimic of biological TEM images and their interpretation.

```
import math
import matplotlib.pyplot as plt
import numpy as np

def triangle(radius, center=(0,0), rotation_deg=90.0):
    cx, cy = center
    angles = [rotation_deg + i*120 for i in range(3)]

    return [(cx + radius *math.cos(math.radians(a)),
             cy + radius *math.sin(math.radians(a))) for a in angles]

def six_angled(radius, center=(0,0), rotation_deg=90.0):
    cx, cy = center
    shorter =radius /math.sqrt(3)
    angles = [rotation_deg + i*60 +30 for i in range(6)]
    return [(cx + shorter *math.cos(math.radians(a)),
             cy + shorter *math.sin(math.radians(a))) for a in angles]

def star_of_david(radius, center=(0,0), rotation_deg=90.0):
    cx, cy = center
    shorter =radius /math.sqrt(3)
    angles = [rotation_deg + i*60 for i in range(6)]
    print(shorter,angles)
    points =[]
    for i in range(6):
        points.append( (cx + radius*math.cos(math.radians(angles[i])),
                cy + radius*math.sin(math.radians(angles[i]))) )
        points.append( (cx + shorter*math.cos(math.radians(angles[i]+30)),
                cy + shorter*math.sin(math.radians(angles[i]+30))) )

    return points

def show_image(img):
    plt.imshow(img, cmap="gray", origin="lower")
    plt.axis("off")
    plt.show()

def plot_polygon_pixelated(points, size=100):
    # Draw polygon edges in a pixelated raster
    from skimage.draw import line

    img = np.zeros((size, size), dtype=np.uint8)

    def to_pix(x, y):
        # Direct mapping: assume coordinates already fit in 0..size-1
        return int(round(x)), int(round(y))

    for i in range(len(points)):
        x0, y0 = to_pix(points[i][0], points[i][1])
        x1, y1 = to_pix(points[(i+1)%len(points)][0], points[(i+1)%len(points)][1])
        rr, cc = line(y0, x0, y1, x1)
        rr = np.clip(rr, 0, size-1); cc = np.clip(cc, 0, size-1)
        img[rr, cc] = 255

    show_image(img)
    return img

def add_gaussian_noise(img, sigma = 400, mean= 0.0):
```

```
    return img + np.random.normal(mean, sigma, img.shape).astype(np.float32)



############### reference proteins images ######################
size =100

# hexagon
six = six_angled(center=(50,50), radius=30.0, rotation_deg=90.0)
im_six = plot_polygon_pixelated(six, size=size)

# star
star = star_of_david(center=(50,50), radius=30.0)
im_star = plot_polygon_pixelated(star, size=size)

# triangle up
tri = triangle(center=(50,50), radius=30.0, rotation_deg=90.0)
im_tri_1 = plot_polygon_pixelated(tri, size=size)

# triangle down
tri2 = triangle(center=(50,50), radius=30.0, rotation_deg=-90.0)
im_tri_2 = plot_polygon_pixelated(tri2, size=size)


# singular noisy protein images
# show_image(add_gaussian_noise(im_star))
# show_image(add_gaussian_noise(im_six))
# show_image(add_gaussian_noise(im_tri_1))
# show_image(add_gaussian_noise(im_tri_2))


############### summed images in hypothesis (stars + triangles) ##########
number_stars =2*96
number_tri_1 =96
number_tri_2 =96
exp1 =np.zeros((size,size))
for i in range(number_stars): exp1 += add_gaussian_noise(im_star)
show_image(exp1)

exp2 =np.zeros((size,size))
for i in range(number_tri_1): exp2 += add_gaussian_noise(im_tri_1)
show_image(exp2)

exp3 =np.zeros((size,size))
for i in range(number_tri_2): exp3 += add_gaussian_noise(im_tri_2)
show_image(exp3)


############### summed images in hypothesis (stars + hexagons) ##########
# As triangles have 2/3 features common to stars, 1/3 features common to hexagons
# they must be in 2/3 cases identified as stars
# in the rest 1/3 cases identified as hexagons

exp1 =np.zeros((size,size))
for i in range(number_stars): exp1 += add_gaussian_noise(im_star)
number_false_stars_1 = int(number_tri_1 *2 /3)
for i in range(number_false_stars_1): exp1 += add_gaussian_noise(im_tri_1)
number_false_stars_2 = int(number_tri_1 *2 /3)
for i in range(number_false_stars_2): exp1 += add_gaussian_noise(im_tri_2)
show_image(exp1)

exp2 =np.zeros((size,size))
```

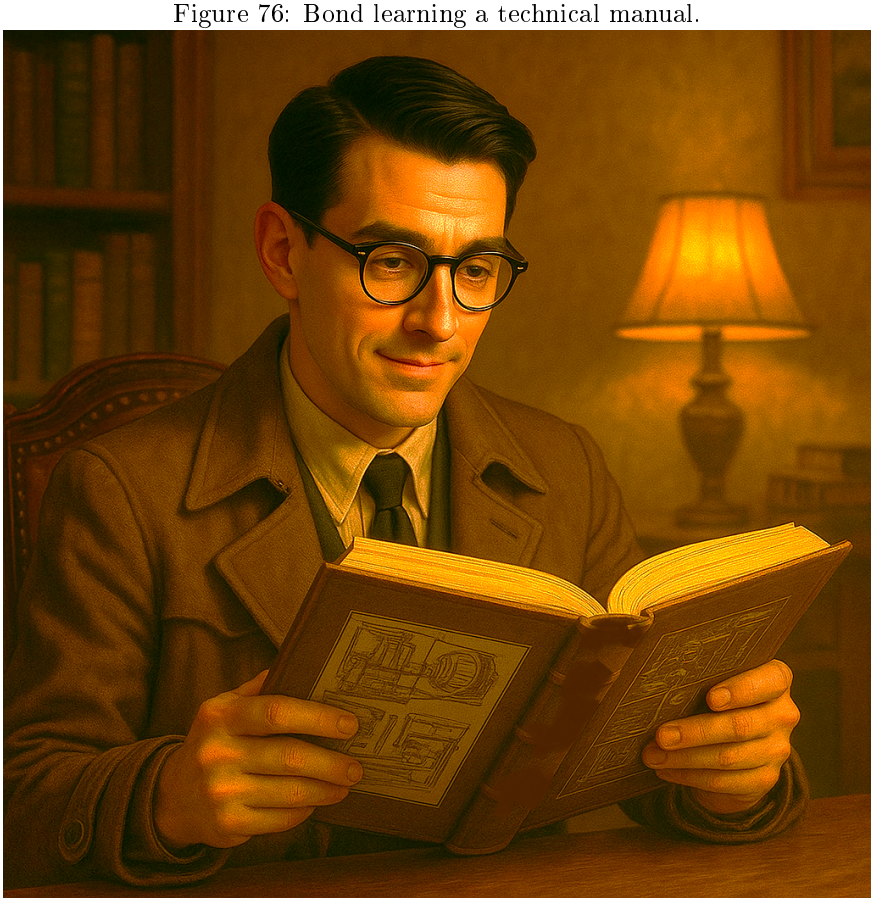
```
number_false_six_1 = int(number_tri_1 *1 /3)
for i in range(number_false_six_1): exp2 += add_gaussian_noise(im_tri_1)
number_false_six_2 = int(number_tri_1 *1 /3)
for i in range(number_false_six_2): exp2 += add_gaussian_noise(im_tri_2)
show_image(exp2)

show_image(exp1+exp2) #an average image
```

12 Mr. Bond and Advanced Machines

12.1 Enchanted Toothbrush

When I visited James Bond next time, I found him buried in a thick book filled with technical drawings and endless specification lists.

Figure 76: Bond learning a technical manual.



"Aha, the annual intelligence-service review for all of Europe has already arrived?" I asked.

"No," he sighed. "This is the manual for an electric toothbrush I bought yesterday. It has five buttons, each of which can be pressed multiple times. That means there are practically infinite operating modes. But I'm halfway through it, and with God's help I might manage to turn the thing on by tomorrow."

"I know you're exceptionally good at handling the advanced innovations your technical department keeps producing," I said, with genuine admiration in my voice.

, "Oh, don't remind me," James groaned. "Every time they service my car, I tremble. They always install some spectacular invention—hovercraft mode, long-range radar, a missile launcher, and God knows what else. All of it powered by control systems nobody understands. I keep telling them: why not just reinforce the engine and improve the steering stability? And for heaven's sake, stop changing my driving interface! Leave your inventions in the background."

"I thought you enjoyed having dozens of options in your car..."

Figure 77: Bond tries to explore a new driving system in his car.

Figure 78: What he does want in his car.

"I enjoy a simple, intuitive interface — one that doesn't require years of training, unlike this damned toothbrush. You know, the very first cars at the end of the 19th century had bizarre controls: levers, knobs, and cranks that varied wildly from model to model. Today's cars are ten times more complex, yet ten times easier to drive. More importantly, the interface is standardized despite fierce competition among manufacturers. I can switch from a Volkswagen to a Ford or a Toyota and start driving immediately. That simplicity emerged partly from long optimization, partly from a common agreement—and it makes life so much easier."

12.2 Simpler Interface

I think everyone would agree with James. I certainly did.

I'd seen countless software packages overloaded with absurdly complicated interfaces, even though I used only a handful of buttons. As a developer myself, I understand the temptation: the desire to provide maximum flexibility, to offer every possible option and parameter. But what developers often overlook is that a user may need only a few simple functions and has no wish to keep every hidden feature in mind. People think differently; what seems intuitive to you might be a puzzle to someone else.

As James put it, designers must aim for such clarity that misunderstanding becomes nearly impossible.

Figure 79: My original interface for the verse generating program.

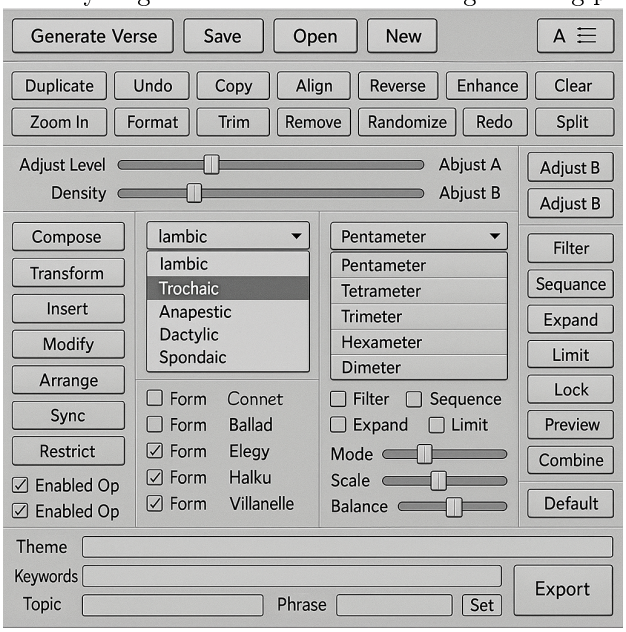
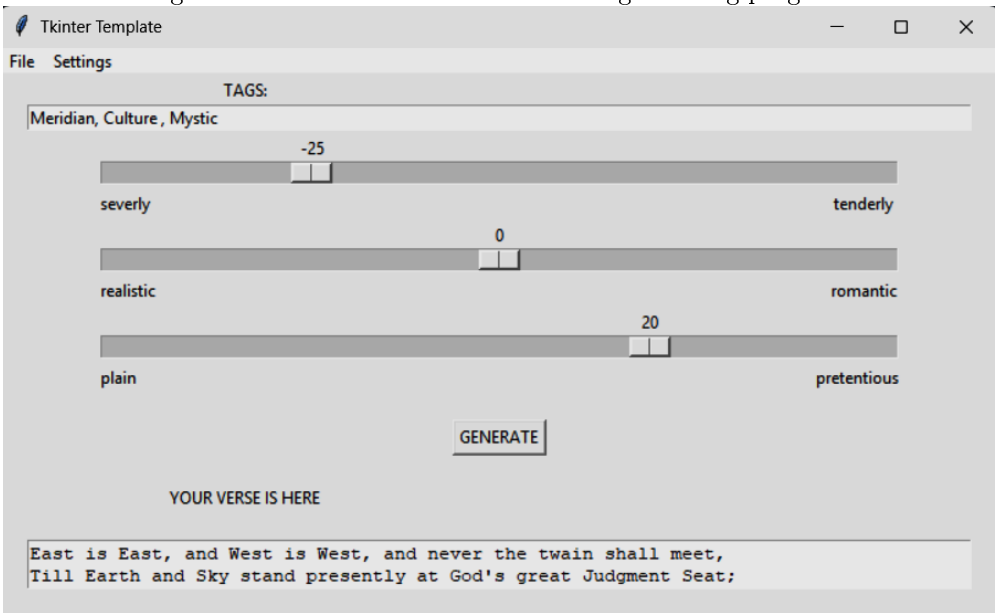


Figure 80: Better interface for the verse generating program.



With these thoughts in mind, I looked at the program I had been developing — a tool that generated verses from a few keywords and tags. Suddenly its interface no longer seemed clear and intuitive, as it had when I constructed it block by block. In fact, if I were not the author, I would have quit the program at

first glance.

After half an hour of work, I arrived at a much simpler control panel. Amazingly, it still supported nearly all the capabilities of the earlier, cluttered interface. Now it looked clean and intuitive — proving once again that beauty is, in this world at least, synonymous with simplicity.

But what about users who want to understand what goes on behind the scenes? Who want to tune the deeper parameters? For them, there is a Settings menu. And those settings are layered, so that each level reflects an appropriate degree of complexity, matching the user's experience.

The conversation with James opened my eyes. It brought my programming style a good deal closer to what people truly need.

12.3 Used code

Listing 23: Interface of the program for writing verses.

```
import tkinter as tk

def generate():
    """
    HERE I SKIP THE ACTUAL VERSE GENERATION CODE
    I DONT WISH THE NUMBER OF POETS TO EXPLODE EXPONENTIALLY
    """
    out.insert("end", "East is East, and West is West, and never the twain shall meet,\nTill
        Earth and Sky stand presently at God's great Judgment Seat;")

def on_change(val,var):
    print(var.name,val)

def dummy():
    print('dummy')

def slider(Quntity, left, right):
    width =180
    Pad =70
    spaces = width - len(left) - len(right)
    label = left + " " * spaces + right

    v = tk.IntVar()
    v.name = Quntity
    s=tk.Scale(root, from_=-50, to=50,
                orient ="horizontal",
                variable =v,
                command =lambda val: on_change(val, v))
    s.pack(fill="x",padx =Pad)
    tk.Label(root, text=label).pack(anchor="w",padx =Pad)

root = tk.Tk()
root.title("Tkinter Template")
PADX =20,
PADY=20
WIDTH=50

# Input tags
tk.Label(text="TAGS:",width=WIDTH).pack(anchor="w")
inp = tk.Entry(root)
inp.insert("end", "Meridian, Culture , Mystic")
inp.pack(fill="x",padx =PADX)

# Sliders
slider('Green','severly','tenderly')
slider('Red','realistic','romantic')
slider('Blue','plain','pretentious')
```

```
# Generate button
tk.Button(root, text="GENERATE", command=generate).pack(pady=PADX)

# Text output
tk.Label(root, text="YOUR VERSE IS HERE",width=50).pack(anchor="w")
out = tk.Text(root, height=2)
out.pack(fill="x",padx =PADX,pady=PADY)

# Menu
menubar = tk.Menu(root)

file_menu = tk.Menu(menubar, tearoff=0)
file_menu.add_command(label="Open", command=dummy)
file_menu.add_command(label="Save", command=dummy)
menubar.add_cascade(label="File", menu=file_menu)

settings_menu = tk.Menu(menubar, tearoff=0)
item1_menu = tk.Menu(settings_menu, tearoff=0)
item1_menu.add_command(label="subItem1", command=dummy)
item1_menu.add_command(label="subItem2", command=dummy)
settings_menu.add_cascade(label="Item1", menu=item1_menu)
settings_menu.add_command(label="Item2", command=dummy)
menubar.add_cascade(label="Settings", menu=settings_menu)

root.config(menu=menubar)


root.mainloop()
```

13 Expanding Dimensions

13.1 Bond Reviews the Bond Stories

After knocking on Bond's door and receiving a slurred permission, I entered and froze on the doorstep. James was deeply absorbed in reading a few pages when I realized, with a jolt, that they were my own stories about him.

Figure 81: Review tends to be infinite.

"Ah, that's you, Mr. Conan-Doyle-Malevich?" he said. "Come in, come in! I am just reviewing your opuses."

Nothing encouraging was written on his face. I felt deeply embarrassed. However, as an experienced reviewer, James began with compliments.

"I appreciate very much that you hid my real name under a pseudonym, James Bond, — a quite ordinary English name that tells people nothing. Although, perhaps, John Smith would have been even better. It is commendable that you used a senseless abbreviation, namely MI-6, to refer to my agency. Its real name is, of course, top secret. But you chose too short an abbreviation. It may eventually coincide with the name of some real company, and then you would be in trouble. You should rather use a generator of random keys and get something like..." — he clicked a few buttons on his device — "for instance, KGBfbiN@S@."

"I tried to do my best..." I mumbled.

"You did, I see. But why do you always portray me with such an idiotic face?" he winced. "And honestly, I find ambiguous statements and inaccurate formulations on every page." He looked at me very skeptically. "Taking into account your very moderate educational background, this is probably excusable."

I uttered not a word.

"However, I cannot tolerate," Bond continued, "when you distort the very meaning of my words. For

Figure 82: Why all reviewers of my papers have such unpleasant faces?

instance, look at our conversation in the bar some years ago (see 5.1). Was it not clear that I meant extending the dimensionality, not squeezing it? You ignored that and used my words to promote your own ideas about automatic reduction of dimensionality in PCA. . . ”

13.2 Patch-based denoising of 2D Images

As always when receiving reviews on my papers, I experienced a sober mixture of shame and rage. Yet I had to admit that James's story about finding a way out of a labyrinth (see 5.1) indeed implied the introduction of an extra dimension, not the reduction of existing ones.

I began to guess where and how Bond might have applied such a strategy and soon found an example. Remember how we denoised 3D spectrum images by finding correlations along their third (spectral) dimension? Then PCA removed much of the noise by clipping components in that dimension. Could we design something similar for ordinary 2D images?

James taught that we must create an extra dimension. For instance, suppose there is a certain correlation among neighbouring pixels in an image. Then, from each pixel, we construct a third dimension consisting of its neighbours. The size of this new dimension depends on how large a neighbourhood we consider.

Does this sound like an unnecessary complication? But if the neighbourhood is correlated, PCA is able to capture the correlation and truncate the third dimension down to the strictly necessary size. When we fold this cube back into a 2D shape, a great deal of random noise disappears. This is called patch-based PCA.

I think I now understand better what James meant when he said that he found a way out of the labyrinth by looking from an extra dimension. Make geometry more complex in order to see a simpler solution! This principle is used in many data science methods, for instance, in Support Vector Machines.

Figure 83: Construct the artificial 3rd dimension from a neighbourhood patch.

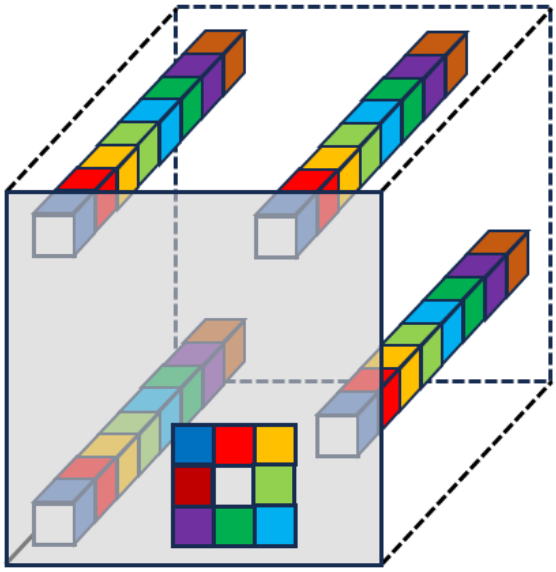
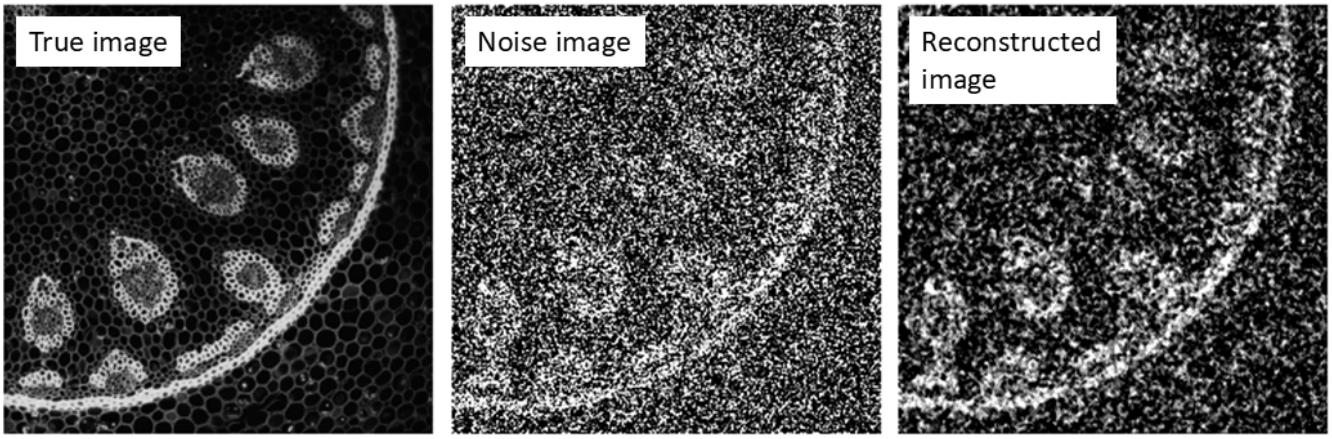


Figure 84: Patch-based PCA removes some fraction of noise from 2D images.



13.3 Used code

Steve:
Denoise looks not significant. I can do same by simple median filering.

Pavel:
You are fully right. Denoising is not as straightforward as that for spectrum-images. However, this example is only an illistration of idea. In reality, patch PCA requires much more efforts like sorting, grouping and optimization of patches which brings us to the domain of dictionary learning. When doing everything correctly, we denoise 2D images without loss of resolution, thus outperform any filtering.

Listing 24: Patch-based PCA.

```
import numpy as np

##############################################
# Patch PCA denoising utilities
##############################################
def extract_patches(img, patch=8, stride=4):
    H, W = img.shape
```

```
    coords, patches = [], []
    for i in range(0, H - patch + 1, stride):
        for j in range(0, W - patch + 1, stride):
            patches.append(img[i:i+patch, j:j+patch].reshape(-1))
            coords.append((i, j))
    return np.stack(patches, axis=0), coords # (N, patch**2), list of (i,j)

def rec_patches(patches_vec, coords, img_shape, patch=8):
    H, W = img_shape
    acc = np.zeros((H, W), dtype=np.float32)
    wgt = np.zeros((H, W), dtype=np.float32)
    for k, (i, j) in enumerate(coords):
        p = patches_vec[k].reshape(patch, patch)
        acc[i:i+patch, j:j+patch] += p
        wgt[i:i+patch, j:j+patch] += 1.0
    return acc / np.maximum(wgt, 1e-8)

def pca_patch_denoise(img_noisy, patch=8, stride=4, left=0.1):
    """
      * stack patches into X with shape (HxW) x (patch**2)
      * center X and make SVD
      * leave only upper components of the 'left' fraction
      * reconstruct patches and overlap-average
    """
    X, coords = extract_patches(img_noisy, patch=patch, stride=stride)

    mu = X.mean(axis=0, keepdims=True)
    Xc = X - mu
    U, S, Vt = np.linalg.svd(Xc, full_matrices=False)

    k =int(left*patch**2)
    Vk = Vt[:k, :]
    scores = Xc @ Vk.T
    Xhat = scores @ Vk + mu

    return rec_patches(Xhat, coords, img_noisy.shape, patch=patch)


################################################
######### Example image ########################
################################################
from skimage import data
from skimage.color import rgb2gray
from skimage.util import img_as_float32

########## Load a flower-related image and convert to grayscale
img_rgb = data.lily()[:, :, :3] # keep RGB, drop alpha
img = img_as_float32(rgb2gray(img_rgb))
mi =np.min(img)
ma =np.max(img)
print('size',img.shape,'min', mi, 'max', ma)

########## Add noise
np.random.seed(0)
sigma = 0.2
noisy = img + sigma * np.random.randn(*img.shape).astype(np.float32)

########## Denoise
left =0.1
denoised =pca_patch_denoise(noisy, patch=8, stride=4, left=left)

########## Plot results ################################
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(11, 4))
def plot_im(ind, img, mi, ma):
    plt.subplot(1, 3, ind)
    plt.imshow(img, cmap="gray",vmin=mi, vmax=ma)
    plt.axis("off")
    plt.tight_layout()

plot_im(1, img, mi, ma)
plot_im(2, noisy, mi, ma)
plot_im(3, denoised, mi, ma)
plt.show()
```

14 Strike Against Extremi(sts)ties

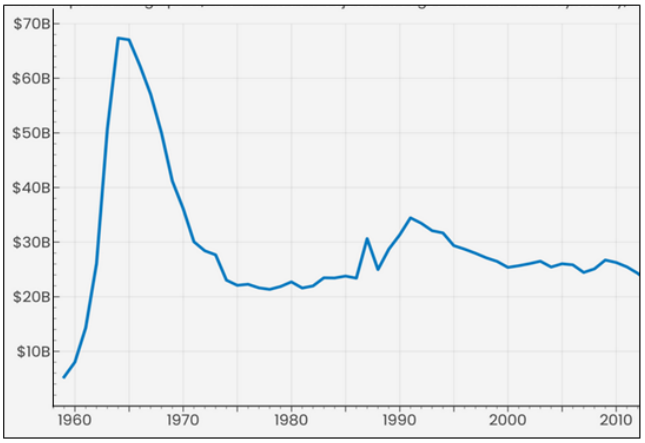
14.1 James Bond and Esoteric Events

James and I were travelling by train to Geneva. We were to participate in the prestigious *International Congress on Espionage & Sabotage*. Time on a train is ideal for leafing through conference materials.

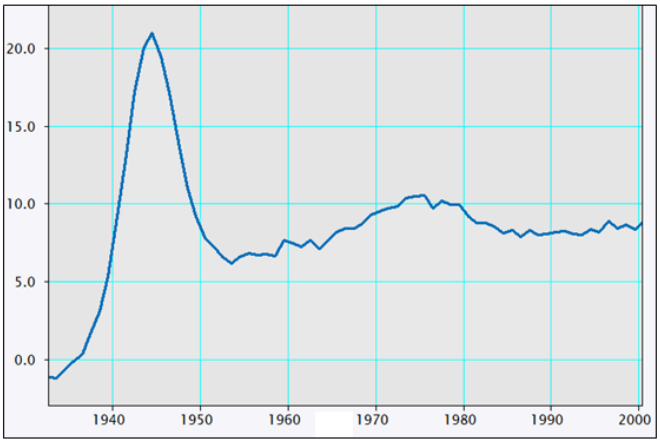
Suddenly, I noticed that James was studying a figure that looked quite familiar to me.

"I didn't know you dealt with spectroscopy," I said. "I see, this is a Strontium L_3 edge as it appears in Electron Energy-Loss Spectroscopy. Wait a moment... I have a similar spectrum, but mine is slightly better calibrated."

Figure 85: left: NASA budget adjusted for inflation (NASA Historical Budget Chart); right: EELS Strontium L_3 edge (Gatan digital version of EELS atlas, C.C. Ahn and O.L. Krivanek).



Annual NASA budget (years)



Sr L_3 edge, background subtracted (eV)

James glanced absent-mindedly at my image and became dumbfounded. A moment later, I observed a rather rare natural phenomenon — astonishment on his face.

"I am not involved in spectroscopy. My graph is the annual budget of NASA, since its inception. But your graph — if you say it is a particular spectrum — is essentially the same..."

However, rational thinking soon prevailed.

"I don't think there is any hidden relation between excitations of atomic shells and the excitation of Americans about space. Most probably, it is just a coincidence," he said. "The number of technical documents is growing exponentially. An enormous variety of combinations appears, and sooner or later even a very improbable parallelism may occur. It's like meteorites. Have you ever observed a meteorite falling?"

"Never."

"Neither have I. And yet, every day several fall on Earth. If we were a community of only a thousand individuals, we would consider such events esoteric and therefore mythological. But we are several billion,

Figure 86: I have never seen such an expression on Bond's face.

and so we know precisely that meteorites exist. Who knows which other rare events would have to be acknowledged as real if our statistics were a million times larger. . . "

He fell silent, lost in thought.

"But such rare events do not affect the general picture. They are washed out after averaging over all available samples," I argued.

"Mostly," James agreed. "But not always. We can easily imagine that a singular, rare, and unbelievable event dramatically changes an entire life and a global trend."

"Black swan?"

"Exactly. By the way, PCA analysis offers a fine illustration of this point. . . " "

14.2 Black Swan of PCA

As there was still half an hour left before the terminal station, we quickly devised an example of a data distribution in which a single outlier completely overrode the PCA trend.

Why did this happen? Because PCA actually finds a line that fits all data points in the sense of minimal least squares. If a singular point lies far away from the others, its coordinates affect the result quadratically.

"It is a pity that Nassim Taleb (The Black Swan: The Impact of the Highly Improbable) is not participating in the congress," James commented. "We could present him with a perfect example of his Black Swan."

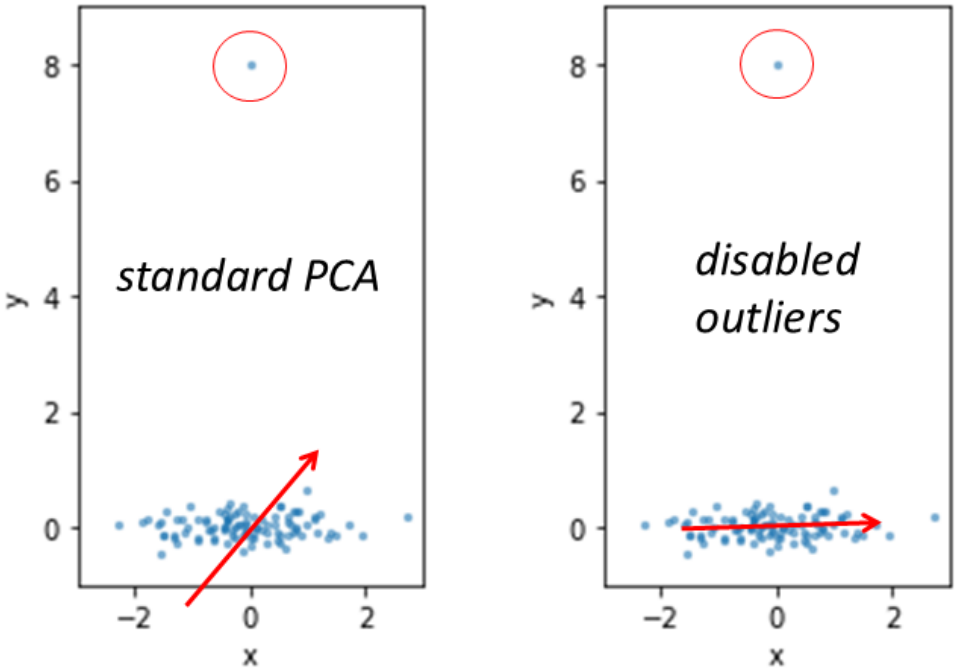
Still, we had no desire to live in Taleb's *Extremistan*, and so we devised a simple workaround. With two additional lines of code, we disabled the influence of extremities. Namely, all data points lying outside a $\pm 4\sigma$ range of the data variance were excluded from the PCA calculation.

We returned to our dull *Mediocristan* and enjoyed life again.

Disclaimer. We did not manipulate the data. We merely defined a reasonable range for calculating the PCA trend. It was, in fact, a very generous range; one would have to be exceptionally reckless to venture beyond it.

Nor did we completely ignore the extreme data point. After projection onto the line defined by the "outlier-robust PCA," it exhibited rather civilized behaviour.

Figure 87: A singular outlier overrides completely the extracted PCA trend (red arrow). Slight modification with disabling extremities restores the reasonable PCA analysis



14.3 Used code

Listing 25: outliers-robust PCA.

```
import numpy as np
import matplotlib.pyplot as plt


def pca_nipals(matrix, Pvec, Sigmas_cut=None):
    for it in range(10): # 10 iterations
        Tvec =np.matmul(matrix,Pvec)
        if Sigmas_cut is not None:
            Sigma =np.sqrt(np.nanvar(Tvec)) #standard deviation in Tvec distribution
            Tvec =np.where(np.absolute(Tvec/Sigma)>Sigmas_cut,0,Tvec) # not account point outside
                +-Sigma_cut*sigma
        Pvec =np.matmul(matrix.transpose(),Tvec)
        Pvec[:,0] =Pvec[:,0]/np.linalg.norm(Pvec) #normalize Pvec to unit length

    return Pvec


# Generate Gaussian-distributed data
N = 100
sigma_x = 1.0 # high variance along x
sigma_y = 0.2 # low variance along y
x = np.random.normal(loc=0.0, scale=sigma_x, size=N)
y = np.random.normal(loc=0.0, scale=sigma_y, size=N)
# PCA TREND IS EXPOCTED TO BE ALONG X. WHILE DISTRIBUTION AROUND Y IS RATHER MINOR NOISE

# add ONE outlier
x[-1] =0
y[-1] =8
```

```
# Combine into a 2D array
data = np.column_stack((x, y)) # shape (N, 2)
print(data.shape)

# Scatter plot: y vs x distribution
fig, ax = plt.subplots(figsize=(8, 4)) # wide figure
ax.scatter(x, y, s=5, alpha=0.5)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim(-3, 3)
ax.set_ylim(-1, 9)
ax.set_aspect('equal', adjustable='box')

# deduce PCA trend (1st PCA component in 2-dimensional space)
Pvec =np.zeros((2,1),np.float32)
Pvec[1,0]=1
Sigmas_cut =4 # variation is considered within +-4 sigma
#Sigmas_cut =None
print(pca_nipals(data, Pvec,Sigmas_cut=Sigmas_cut))
```

Anonymus:

Pavel, do you really believe in all black-swan-garbage of this Gauss hater- Taleb?

Pavel:

Funny enough, I posed to James exactly the same question in the train.

"**I? I am** a follower of Nassim Taleb ???' -asked he surprised and then smiled sarcastically –
'I am. Especially, I like his amazing discovery that **'Not every distribution is a Gaussian and not every Gaussian is a distribution...'**"