# Biological Imaging Systems: Image Analysis and Processing

**Project: Robust Principal Component Analysis**
**Student Registration №: 1120038**
**Full Name: Kristiani Thimo**

## Introduction

**Robust Principal Component Analysis** (**RPCA**) is a modification of PCA which works well with respect to ***grossly*** **corrupted** observations [1].

A number of different approaches exist for Robust PCA, including an idealized version of Robust PCA, which aims to recover **a low-rank matrix $L_0$** from highly corrupted measurements **$M = L_0 + S_0$** [1].

This decomposition in low-rank and sparse matrices can be achieved by techniques such as Principal Component Pursuit method (PCP), Stable PCP, Quantized PCP, Block based PCP and Local PCP [1].

Then, some optimization methods are used [1] such as:

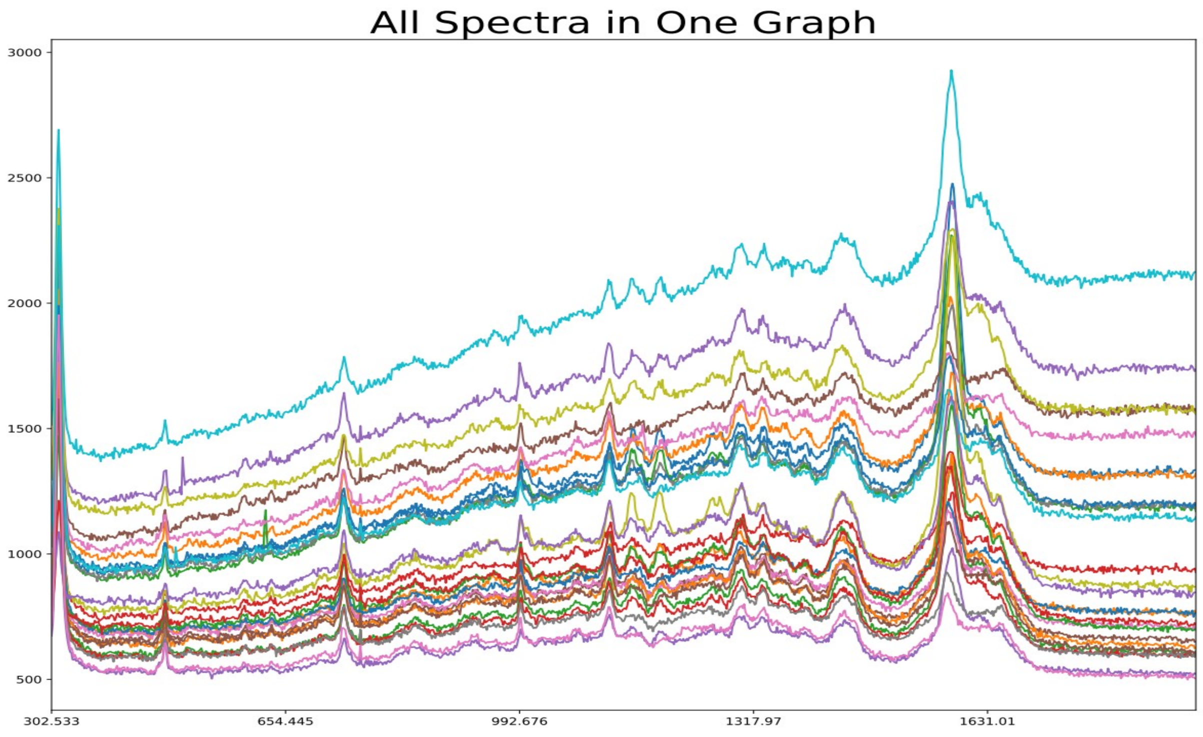
- Augmented Lagrange Multiplier Method (ALM)
- Iteratively Reweighted Least Squares (IRLS)
- Alternating Direction Method (ADM)
- Fast Alternating Minimization (FAM)

## Usage, Results and Discussion

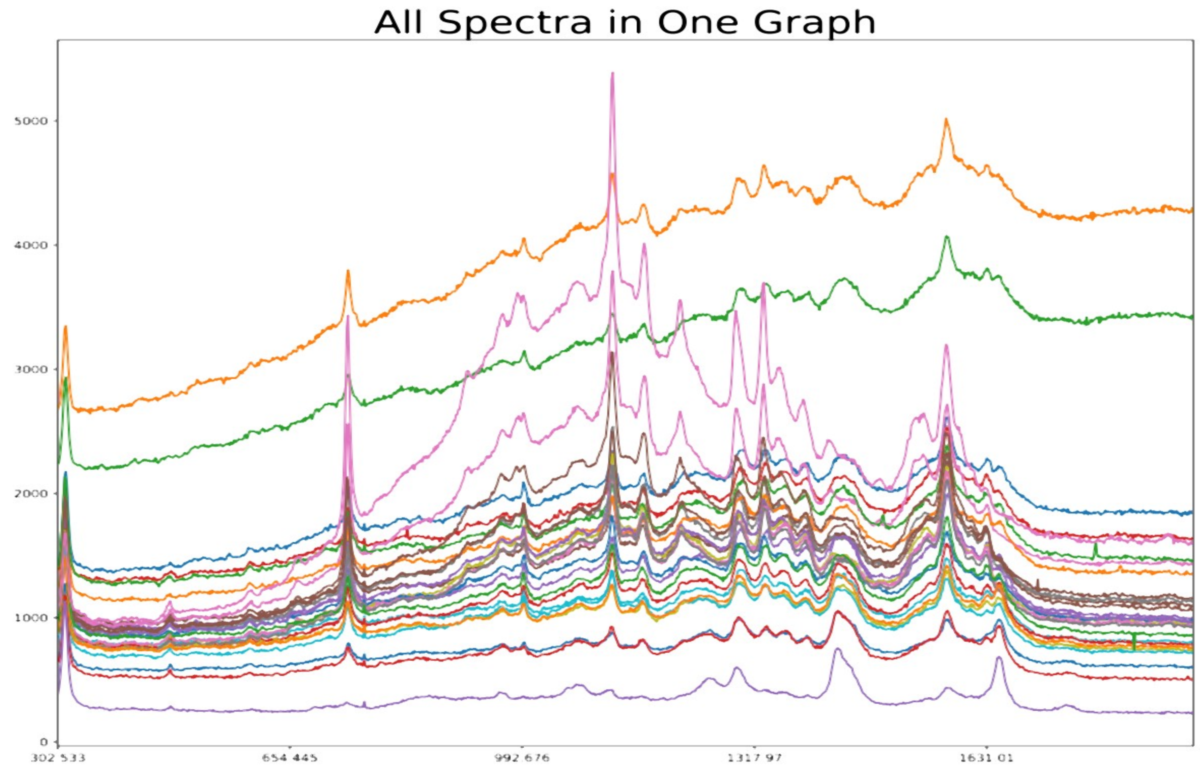
### Data Differentiation

Data (matrices) that cannot be differentiated (graphs comparison) can be differentiated after applying RPCA to them (graphs of their Sparse matrices comparisons). That happens because Sparse can be considered noise (which is supposed to be distributed sparsely among the data), whilst Low Rank (underlying less noisy) is what is the constant part of the matrix [4] [5].

A PCP algorithm via alternating directions [2] [3], performed to data from liver tissues, a malignant and a normal one.



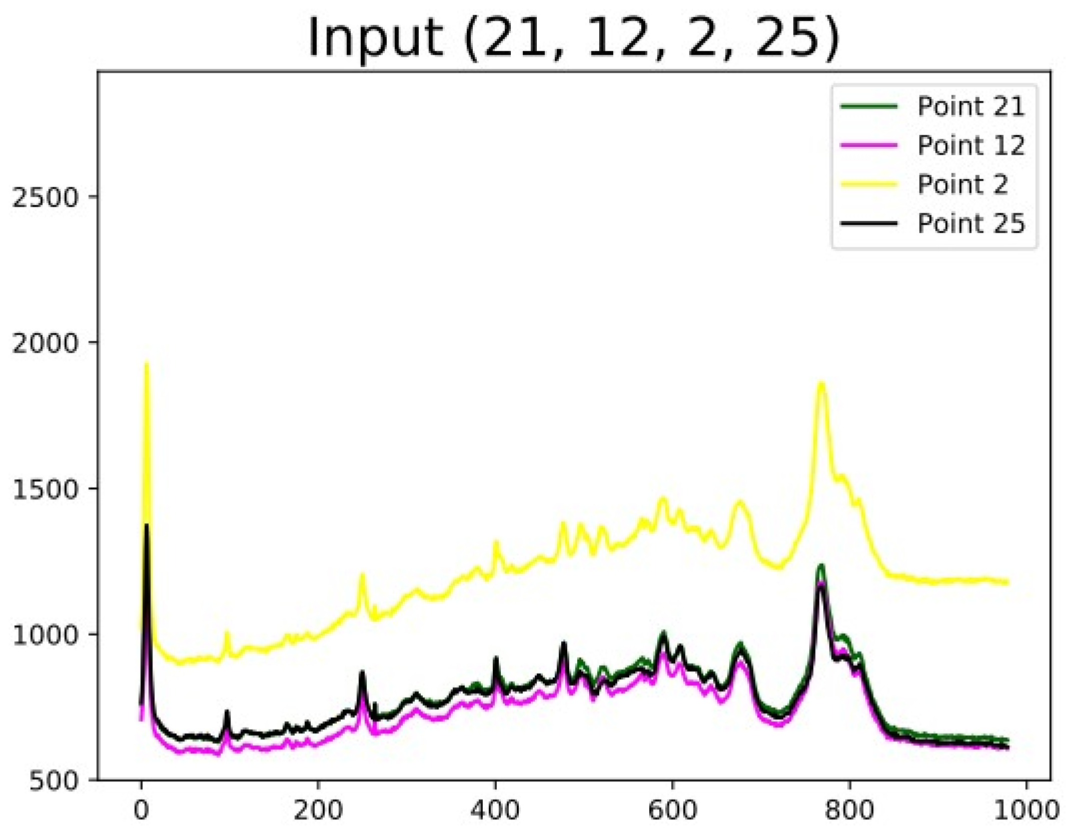
[Figure 1]: Plot of the initial data (spectra) from the normal liver tissue.



[Figure 2]: Plot of the initial data (spectra) from the malignant liver tissue.

Figures 1 and 2 show plots from the initial data: the data are separable, as regards to my eyes, but the separation is not clear.

Sparse comparisons will show much more clearly that the data differentiate.

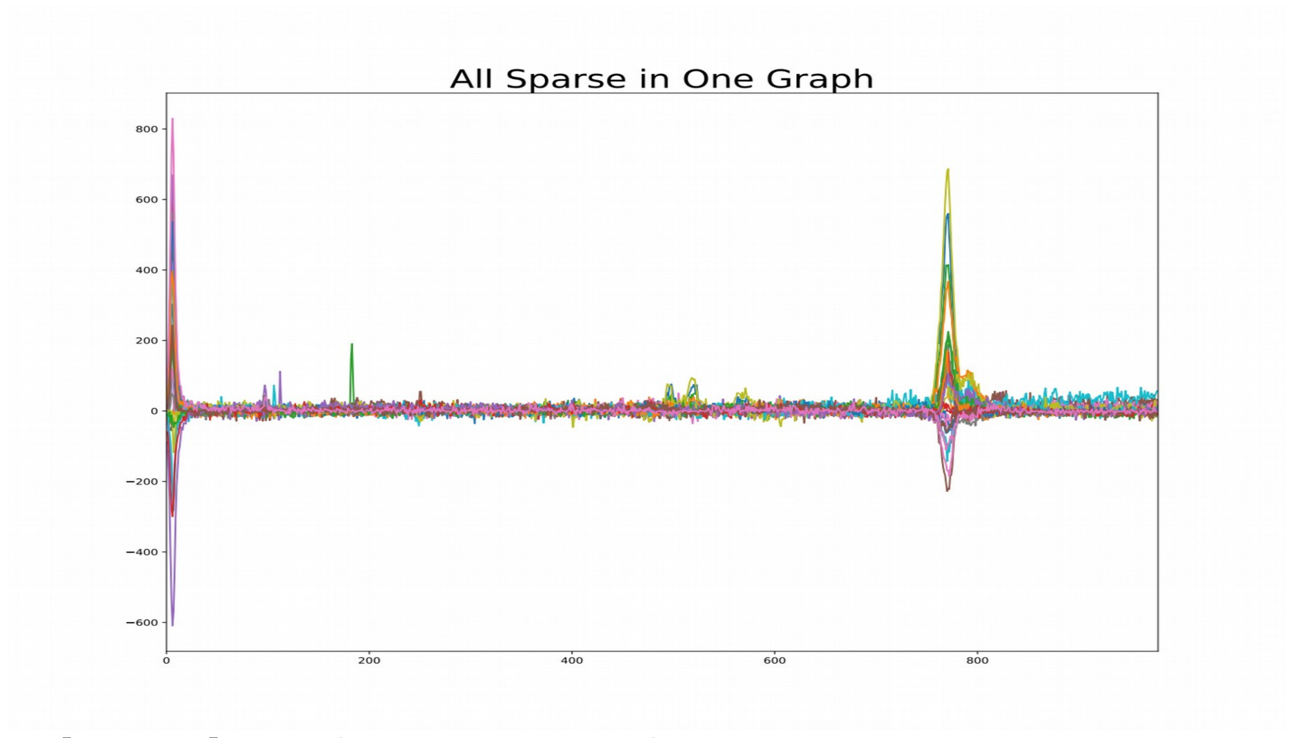


[Figure 3] Low Rank of four random points out of 27 in the normal data.

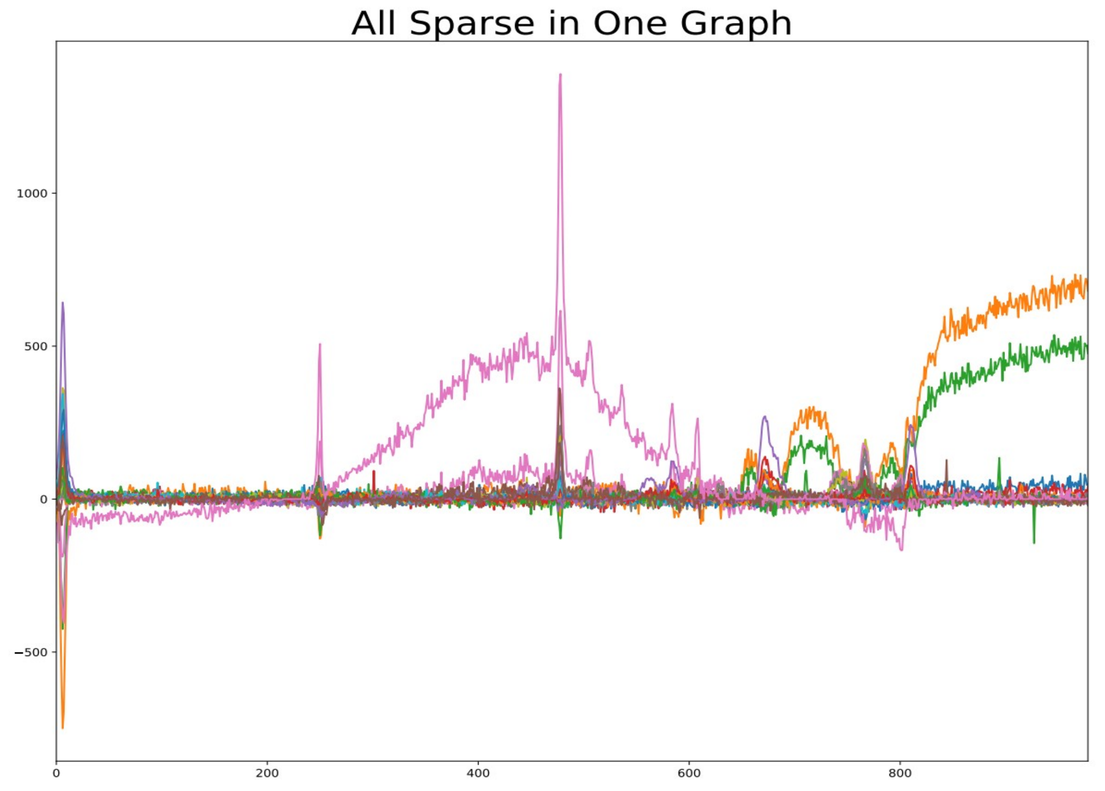
Figure 3 shows the Low Rank plots of four randomly chosen points out of the 27 in the normal data: the plots look the same but translocated (different intensities), which is not the case at the same degree when plotting the Low Ranks of four random points in the malignant data (figure not included here, but there is one in the Supplementary Figures).

Figures 4 and 5 below show the graphs of all Sparse part for every point out of 27 for the normal and malignant liver data accordingly.

Undoubtedly, the separation between them is now crystal clear.



[Figure 4] Plot of the Sparse part of all 27 points in the normal data.



[Figure 5] Plot of the Sparse part of all 27 points in the malignant data.

All results, including plots of Low Rank plots of all points and separate plots of each point for both data, are included as Supplementary Figures.

**RPCA for Dimensionality Reduction**

**Method 1**

Apart from telling us where two matrices of data differ, another usage of RPCA is that it can be used as a noise reduction transformation before the usage of a PCA analysis for dimensionality reduction [5].

First, RPCA is applied, then PCA in the L matrix result of the RPCA performed.

The implementation is a function that requires the file and what percentage of variance (greater than) needed, and it was executed for 95% variance in the normal liver dataset and the result was that 2 principal components suffice.

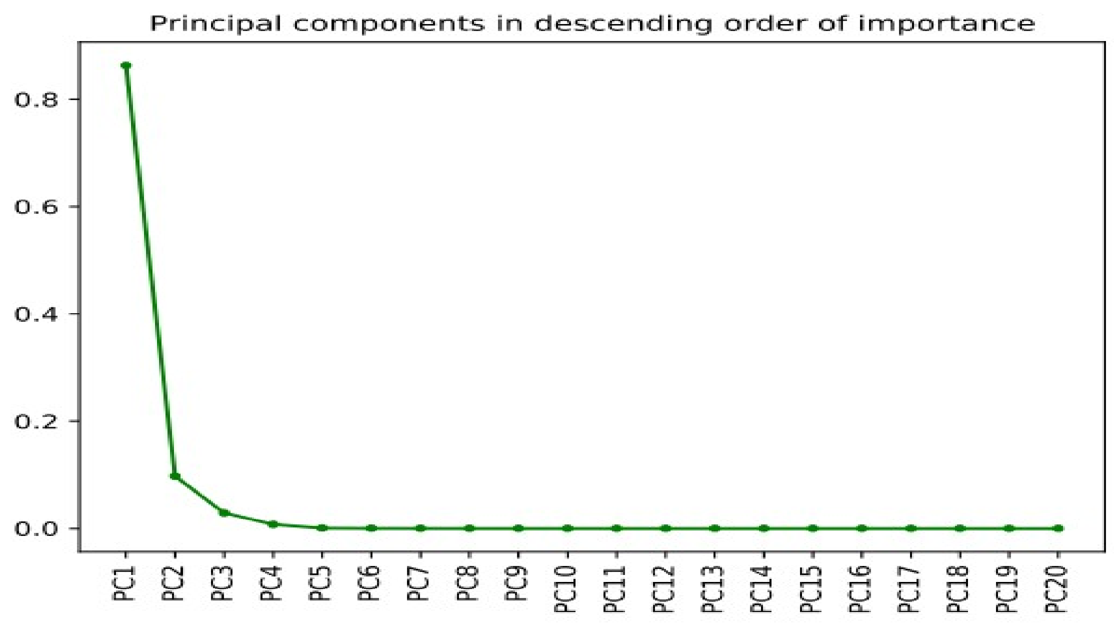
The call made is included in the files attached.

**Method 2**

Same procedure to Method 1, but this time using the RPCA implementation of **sklearn**, which seems to be designed and implemented to achieve dimensionality reduction; it actually has an argument option (n_components = None) when calling where the number of component stays the same as the dimension, but it will raise errors (due to infeasible mathematical operations involved) when fitting and transforming, and so it could be done to not lead to dimensionality reduction only by changing the source code.

Executed with same inputs to Method 1 and had the same output (i.e. 2).

Figure below (the elbow method graph) shows variance explained for the 20 first principal components (it's a plot result of the calls).



## RPCA for Matrix Completion

Another usage of RPCA is **Matrix Completion** (when values missing) [2], which won't be covered here.

# Technical Report

- The implementation is in **Python** (version 3) programming language.

- For more details, check the source code: the documentations and the comments are verbose enough, i.e. I deem they are sufficient enough, to cover anything related to usage and implementation, in all details.

How to run the codes (for Unix-like terminals):

```
python3 main_norm.py # when being in the main_norm directory
python3 main_mal.py # when being in the main_mal directory

python3 method1.py # in the Method1_no_sklearn_RPCA directory
python3 method2.py # in the Method2_sklearn_RPCA directory
```

The RPCA implementations used can be found in the links below:

1st: https://github.com/dganguli/robust-pca/blob/master/r_pca.py (PCP)

2nd: https://bsharchilev.github.io/RobustPCA/_modules/rpca/m_est_rpca.html

However, there are enough alternatives that could be used instead, the recommended ones are:

- https://gist.github.com/bmcfee/a378bfe31a75769c583e#file-rpca-scikit-learn-ipynb (includes usage examples)

- https://github.com/nwbirnie/rpca/blob/master/rpca.py (ALM Method)

- https://github.com/jkarnows/rpcaADMM/blob/master/rpcaADMM.py (RPCA via ADMM method; the implementation uses **multiprocessing** methods and leads to a huge speedup gain)

The last one found and described in the following link:
https://jeremykarnowski.wordpress.com/2015/08/31/robust-principal-component-analysis-via-admm-in-python/
which sums up and has hyperlinks to most in Python RPCA implemenations.

## References

All links not included here, but included as comments in source codes and all links used before the References part in this report.

Also, all PDFs attached in the **PDF_References** file.

[1] RPCA: https://en.wikipedia.org/wiki/Robust_principal_component_analysis
[2] Algorithms: https://www.comp.nus.edu.sg/~cs5240/lecture/robust-pca.pdf
[3] PCP Method: https://arxiv.org/abs/0912.3599
[4] RPCA: https://kojinoshiba.com/robust-pca/
[5] https://stackoverflow.com/questions/40721260/how-to-use-robust-pca-output-as-principal-component-eigenvectors-from-traditio
[6] https://bsharchilev.github.io/RobustPCA/rpca.html
[7] https://github.com/bsharchilev/RobustPCA
[8] https://github.com/dganguli/robust-pca