

Problem 1. *PCA on a real data set*

Solution. This problem set tasks us with performing a Principal Components Analysis on a data set of central optical spectra of 9,713 nearby galaxies from the Sloan Digital Sky Survey. We will tackle various subsections of this problem which trace out the steps of performing a PCA.

a) Firstly, we read the .fits file in using Astropy and by following the steps outlined in the problem description. I then plotted the spectra of 5 galaxies from a range of data available. I then overlaid the wavelengths of emissions corresponding with the Balmer series since they lie in the same region as our data. Figure 1 shows that some of the peaks of our plotted galaxies align with the transitions in the Hydrogen atom.

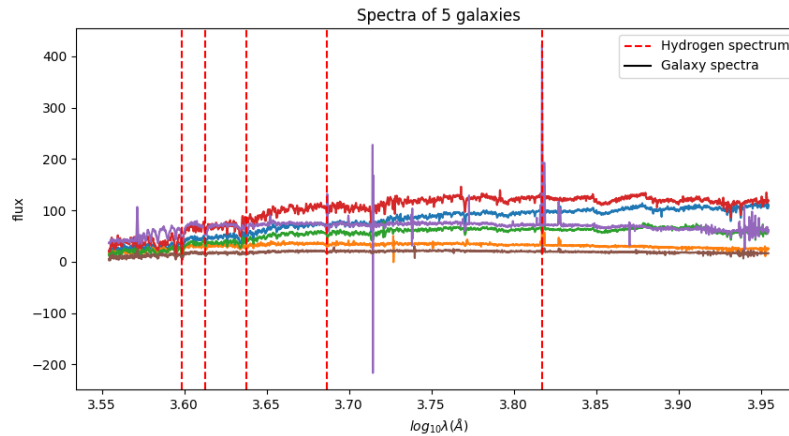


Figure 1: Comparing Spectra of Galaxies with Balmer series

b) Part b tasks us with normalizing the data. We need to normalize the data in a way that the integrals over wavelength are the same since this ensures numerical stability and facilitates our PCA. As per the question, it is close enough to sum the values over the wavelength for each galaxy. So, I calculated the normalization constant as a sum of fluxes for each row and stored it as a $N_{galaxy} \times 1$ array.

c) For part c, I calculated the residuals as follows:

$$\vec{r}_i = \vec{f}_i - \vec{f}_m$$

This keeps the residuals of all the galaxies i varying around zero. This ensures that the PCA does not spend an eigenvector to explain the mean offset from zero. With our data now normalized and centered around the origin, we can proceed with the PCA.

d) For part d, we commence by calculating the covariance matrix. For a residual matrix \mathbf{R} , we can calculate the covariance matrix \mathbf{C} as the dot product of \mathbf{R} and its transpose.

I first cast the residuals as a matrix \mathbf{R} with dimensions $N_{galaxies} \times N_{waves}$. Since the covariance matrix needs to be of dimension $N_{waves} \times N_{waves}$, the covariance matrix is calculated as follows:

$$\mathbf{C} = \mathbf{R}^T \cdot \mathbf{R}$$

After calculating the covariance matrix, I calculated the eigenvalues and eigenvectors using Numpy's `linalg.eig()` function. I then sorted the eigenvectors in descending order by its corresponding eigenvalue. Plotting the first five eigenvectors gives us Figure 2.

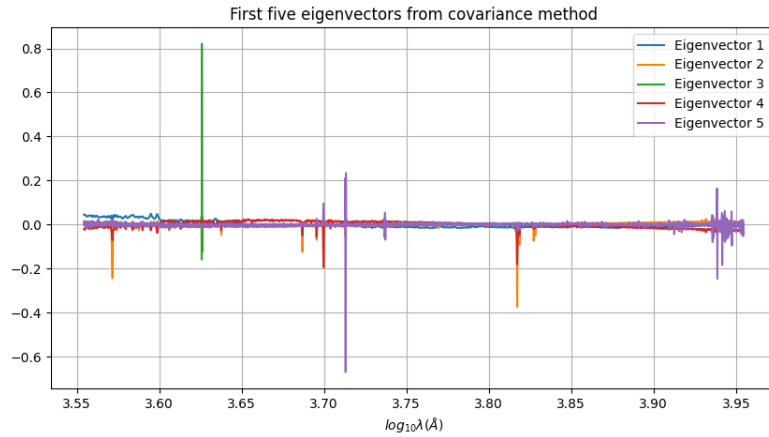


Figure 2: Eigenvectors from covariance matrix

e) Now, we perform a PCA again but this time using Singular Value Decomposition (SVD).

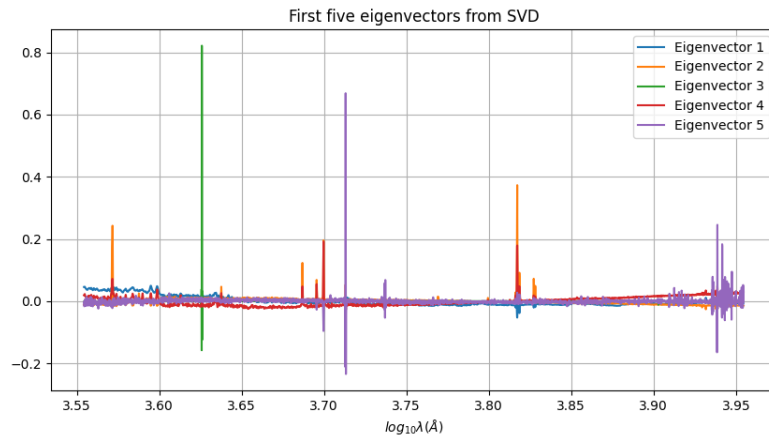


Figure 3: Eigenvectors from SVD

Figure 3 has identical features to the eigenvectors obtained from the covariance method, however most features seem to be reflected on the $y=0$ axis. Comparing the eigenvectors from the two methods in Figure 4 shows that some of the SVD values are reflected across the $x=0$ axis.

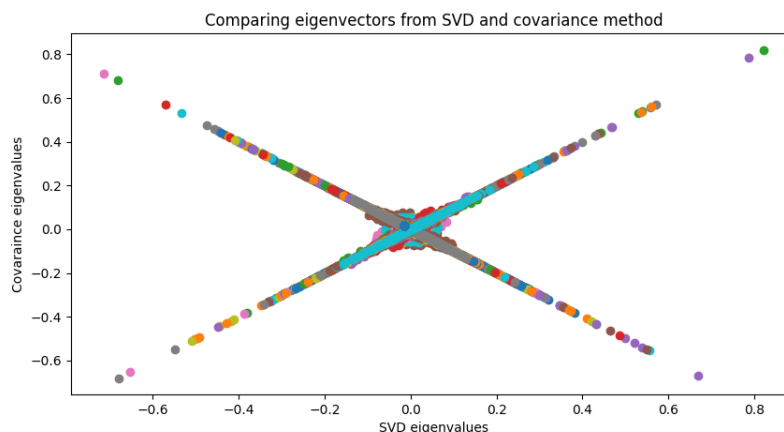


Figure 4: Comparing eigenvectors from the two methods

This is because the right singular vectors from the SVD are equivalent (up to a sign flip) to the eigenvectors obtained from the covariance matrix method. Eigenvectors calculated using software packages might have signs that depend on the specific algorithm used by the software. However, eigenvectors can be flipped in sign but will still span the same subspace. Hence the eigenvectors from both methods are equivalent!

I then compared the times taken for both process (Figure 5). R was first type cast as a matrix for the Covariant calculation. I recorded the time taken to calculate the covariance matrix C from R and to calculate the eigenvectors using Numpy's `linalg.eig()` function. For the SVD method, I recorded the time taken to get the eigenvectors using Numpy's `linalg.svd()` function and passing the residuals array as a parameter. The SVD method took slightly longer than calculating the covariance for our dataset.

```
Time for covariance method: 27.237760066986084
Time for SVD method: 34.12762403488159
```

Figure 5: Comparing execution times for the two methods

f) Since the SVD method took longer for our data set, calculating the covariance matrix seems to be a better method. However, let's compare the two methods in a more general context.

Looking at the condition numbers in Figure 6, we can see that R has a condition number of $6.5 * 10^6$ and C 's is $2.7 * 10^{10}$. This make sense since the condition number of C should be the square of R . R is already high to begin with, which tells us that it is ill-conditioned.

```
Condition number of R: 6561841.5
Condition number of C: 26954813000.0
```

Figure 6: Comparing condition numbers

Calculating the covariance leads to a matrix with an even bigger condition number whose numerical solutions are unstable and may be unreliable. This is an ideal example of where using SVD would be more numerically stable and hence a better choice than constructing the covariance matrix.

SVD might also be a better choice when the matrix is sparse since we can utilize SciPy's `sparse` package for efficient computation.

g) Now, we perform PCA dimensionality reduction by first projecting the residuals `R` to the first `N` eigenvectors to get a set of coefficients `coeffs` where,

$$\text{coeffs}_{ij} = R_i \cdot \text{eigenvecs}_j$$

where, coeffs_{ij} is the dot product of the residuals of the i^{th} galaxy and the j^{th} largest eigenvector. In my program, this is calculated by using the `np.dot()` function with the residuals and eigenvectors. These coefficients are the weights of each eigenvector. We can approximate the original spectra using the first `N` principal components with the formula:

$$\tilde{f}_i = (f_m + \text{coeffs}_{ij} \cdot \text{eigenvecs}_j) \times \text{normalization_constant}$$

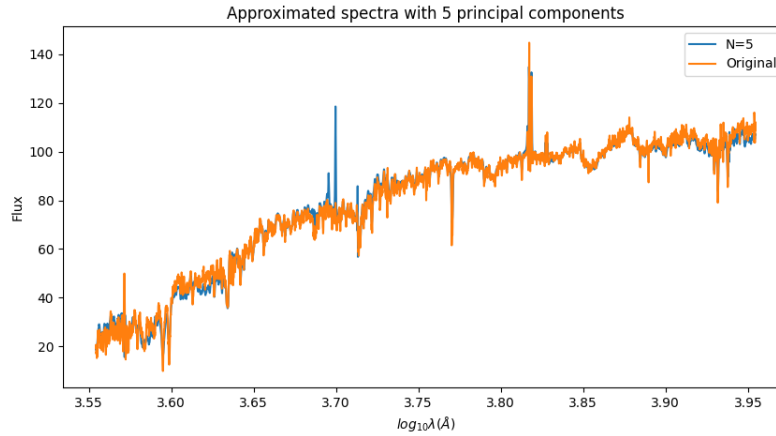


Figure 7: Approximate spectra of the first galaxy in our dataset

Approximating the first spectra using the first 5 principal components gives us the plot in Figure 7. Our predicted curve fits the general trend of the data but fails to capture some features. However, it is a very close approximation. Taking more principal components would give us an even more accurate representation of the original data.

h) We will now plot and compare the coefficients related to the first, second and third principal components. As seen in Figure 8, most of the points in the c_0 vs c_1 graph lie along the c_0 axis but have some spread along c_1 . The points in the c_0 vs c_2 graph are horizontal along c_0 with almost no spread in the c_2 axis. This suggests that the first principal component captures most of the variance in the data and c_0 is the direction of maximum variance. The spread in the c_1 axis suggests that c_1 captures the variance missed by c_0 .

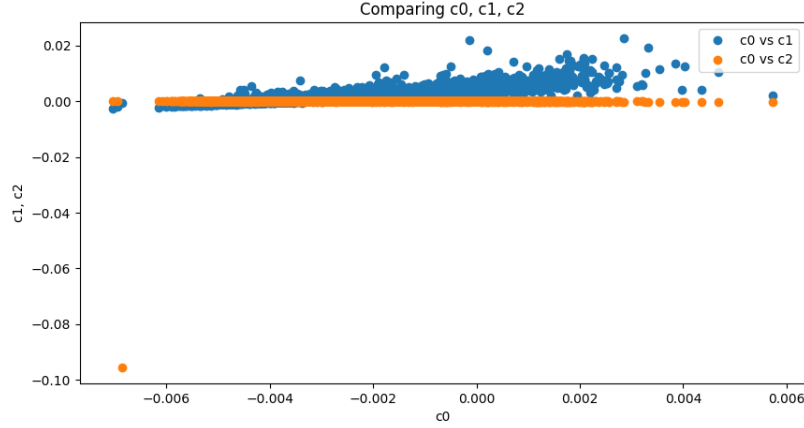


Figure 8: c_0 vs c_1 and c_2

i) For the last part of the problem, we calculate the squared fractional residuals between the original spectra and the approximate spectra reconstructed using the first N_c principal components where we will vary $N_c = 1, 2, \dots, 20$. Plotting the squared fractional residue as a function of N_c gives us the graph in Figure 9.

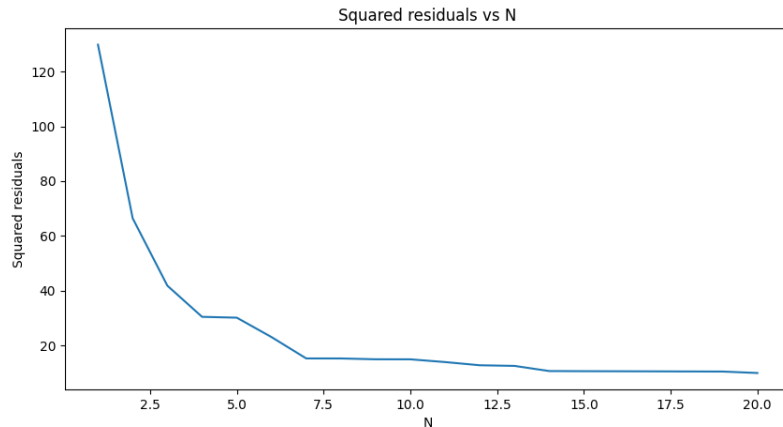
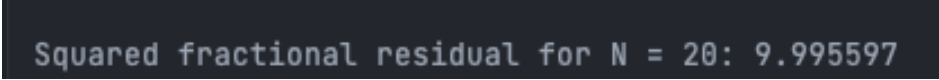


Figure 9: Squared fractional residuals

The squared fractional residual is initially high when we only take a few principal components, starting with a residual of around 120 for $N_c=1$. As we increase the number of

principal components, the residuals start decreasing, which indicates a better approximation of the spectra. The squared fractional residual for $N_c = 20$ was 9.995597 which is still quite high. We could get smaller residuals by taking a higher value of N_c .



```
Squared fractional residual for N = 20: 9.995597
```

Figure 10: Squared fractional residual for $N_c=20$