1. Part A – Optimisation

A furniture company will sell three types of chairs to UQ (types A, B, and C), all made from the same wood material. Suppose a linear equation is to be fit to predict the wood material price as a function of the quantity of these three types of chairs sold, given the following:

Table 1 – Number of each chair type and respective wood prices

Number of Type A	Number of Type B	Number of Type C	Price of Wood
90	10	80	50
100	80	140	20
170	30	160	90
169	31	155	100
100	90	90	40

Assume the prediction equation is $p(n) = ax(n) + by(n) + cz(n)$, where a, b, c are the prediction parameters for the quantities of chair types A, B and C to be sold, respectively. Define $x(n)$, $y(n)$, $z(n)$ as the observations on the number of types A, B, and C sold, respectively (see columns 1, 2, 3 in Table 1), and $p(n)$ as the observed price (see column 4 in Table 1). n identifies the observation index.

Please develop a Linear Programming (LP) model to minimise:

a) **sum of the absolute deviations**
b) **largest absolute deviation**
c) **sum of the squared deviations**

Given the above criteria, let "N" be the matrix representing the observed number for each type of chair and "p" be the vector representing the price of wood for each observation.

$$\boldsymbol{N} = \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix}, \quad \boldsymbol{p} = \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

1.1. Minimise Sum of Absolute Deviations

Consider the mathematical expression for the sum of absolute deviations:

$$\sum \left| p_{observed} - p_{predicted} \right|$$

where:

$$p_{observed} = p(n)\,, \;\; p_{predicted} = a \cdot x(n) + b \cdot y(n) + c \cdot z(n)$$

Therefore, the mathematical expression for the sum of absolute deviations can now be rewritten in the following form:

$$\sum | \, p(n) - (a \cdot x(n) + b \cdot y(n) + c \cdot z(n)) \, |$$

Additionally, to further simplify the problem, an auxiliary vector can be introduced to represent the absolute deviations, thus removing the absolute value expression such that:

$$p(n) - (a \cdot x(n) + b \cdot y(n) + c \cdot z(n)) \leq s(n)$$

$$\big(a \cdot x(n) + b \cdot y(n) + c \cdot z(n)\big) - p(n) \leq s(n)$$

This simplifies the problem because minimising all elements in the auxiliary vector will naturally minimise all absolute deviations for each observation. The above inequality expressions can also be written in matrix form in preparation for use with the MATLAB linprog function:

$$\boldsymbol{p} - [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T \leq \boldsymbol{s}$$

$$[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{p} \leq \boldsymbol{s}$$

where:

$$\boldsymbol{p} = p(1), p(2), \dots, p(n)\,, \;\; \boldsymbol{s} = s(1), s(2), \dots, s(n)$$

$$\boldsymbol{a} = a \cdot x(1), a \cdot x(2), \dots, a \cdot x(n)$$

$$\boldsymbol{b} = b \cdot y(1), b \cdot y(2), \dots, b \cdot y(n)$$

$$\boldsymbol{c} = c \cdot z(1), c \cdot z(2), \dots, c \cdot z(n)$$

The above expressions can be rearranged as follows:

$$-[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{s} \leq -\boldsymbol{p}$$

$$[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{s} \leq \boldsymbol{p}$$

This set of expressions can now be transformed into the form $\boldsymbol{A} \cdot \boldsymbol{X} \leq \boldsymbol{B}$ required by the linprog, where:

$$\boldsymbol{A} \cdot \boldsymbol{X} = \begin{bmatrix} -[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{s} \\ [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{s} \end{bmatrix}, \;\; \boldsymbol{B} = \begin{bmatrix} -\boldsymbol{p} \\ \boldsymbol{p} \end{bmatrix}$$

Finally, the expressions for matrices $\boldsymbol{A}$, $\boldsymbol{X}$ and $\boldsymbol{B}$ can be defined as follows:

$$\boldsymbol{A} = \left(\begin{array}{ccc|ccccc} -x(1) & -y(1) & -z(1) & -1 & 0 & 0 & 0 & 0 \\ -x(2) & -y(2) & -z(2) & 0 & -1 & 0 & 0 & 0 \\ -x(3) & -y(3) & -z(3) & 0 & 0 & -1 & 0 & 0 \\ -x(4) & -y(4) & -z(4) & 0 & 0 & 0 & -1 & 0 \\ -x(5) & -y(5) & -z(5) & 0 & 0 & 0 & 0 & -1 \\ x(1) & y(1) & z(1) & -1 & 0 & 0 & 0 & 0 \\ x(2) & y(2) & z(2) & 0 & -1 & 0 & 0 & 0 \\ x(3) & y(3) & z(3) & 0 & 0 & -1 & 0 & 0 \\ x(4) & y(4) & z(4) & 0 & 0 & 0 & -1 & 0 \\ x(5) & y(5) & z(5) & 0 & 0 & 0 & 0 & -1 \end{array}\right), \quad \boldsymbol{X} = \begin{pmatrix} a \\ b \\ c \\ s(1) \\ s(2) \\ s(3) \\ s(4) \\ s(5) \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} -p(1) \\ -p(2) \\ -p(3) \\ -p(4) \\ -p(5) \\ p(1) \\ p(2) \\ p(3) \\ p(4) \\ p(5) \end{pmatrix}$$

Furthermore, the $\boldsymbol{f}$ vector can be defined to minimise the auxiliary variables which were introduced to represent the absolute deviations. By minimising all absolute deviations, the sum of absolute deviations will also be minimised.

$$\boldsymbol{f} = (0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$$

Given the above matrices, linprog can now be used. The function can be called as follows:

```
X = linprog(f, A, B)
```

The resultant $\boldsymbol{X}$ vector will contain the optimal solutions for the variables $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ respectively, in addition to the results for the auxiliary variables. The auxiliary variable results can be ignored since they are just placeholders to simplify the optimisation process. The results from running the linprog function are as follows:

$$\boldsymbol{X} = \begin{pmatrix} a \\ b \\ c \\ s(1) \\ s(2) \\ s(3) \\ s(4) \\ s(5) \end{pmatrix} = \begin{pmatrix} 0.9803 \\ -0.2039 \\ -0.4408 \\ 0.9211 \\ 0 \\ 0 \\ 8.9803 \\ 0 \end{pmatrix}, \quad \begin{cases} a = 0.9803 \\ b = -0.2039 \\ c = -0.4408 \end{cases}$$

With the above parameters, the predicted price of material for each observation can now be evaluated as follows:

$$p_{predicted}(n) = 0.9803 \cdot x(n) - 0.2039 \cdot y(n) - 0.4408 \cdot z(n).$$

$$p_{predicted} = \boldsymbol{N} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \cdot \begin{bmatrix} 0.9803 \\ -0.2039 \\ -0.4408 \end{bmatrix} = \begin{bmatrix} 50.9211 \\ 20 \\ 90 \\ 91.0197 \\ 40 \end{bmatrix}$$

1.2. Minimise Largest Absolute Deviation

Consider the mathematical expression for the largest absolute deviation:

$$\max\left(\left|p_{observed} - p_{predicted}\right|\right)$$

where:

$$p_{observed} = p(n)\,,\;\; p_{predicted} = a \cdot x(n) + b \cdot y(n) + c \cdot z(n)$$

Therefore, the mathematical expression for the largest absolute deviation can now be rewritten in the following form:

$$\max\left(\,\left|\,p(n) - \left(a \cdot x(n) + b \cdot y(n) + c \cdot z(n)\right)\,\right|\,\right)$$

Additionally, to further simplify the problem, an auxiliary vector can be introduced to represent the maximum absolute deviation, such that:

$$p(n) - (a \cdot x(n) + b \cdot y(n) + c \cdot z(n)) \leq s$$

$$\left(a \cdot x(n) + b \cdot y(n) + c \cdot z(n)\right) - p(n) \leq s$$

This simplifies the problem because minimising the auxiliary variables will naturally minimise the maximum absolute deviations for each observation. The above inequality expressions can also be written in matrix form in preparation for use with the MATLAB linprog function:

$$\boldsymbol{p} - [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T \leq s$$

$$[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - \boldsymbol{p} \leq s$$

where:

$$\boldsymbol{p} = p(1), p(2), \ldots, p(n)$$

$$\boldsymbol{a} = a \cdot x(1), a \cdot x(2), \ldots, a \cdot x(n)$$

$$\boldsymbol{b} = b \cdot y(1), b \cdot y(2), \ldots, b \cdot y(n)$$

$$\boldsymbol{c} = c \cdot z(1), c \cdot z(2), \ldots, c \cdot z(n)$$

The above expressions can be rearranged as follows:

$$-[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - s \leq -\boldsymbol{p}$$

$$[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - s \leq \boldsymbol{p}$$

This set of expressions can now be transformed into the form $\boldsymbol{A} \cdot \boldsymbol{X} \leq \boldsymbol{B}$ required by the linprog, where:

$$\boldsymbol{A} \cdot \boldsymbol{X} = \begin{bmatrix} -[\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - s \\ [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]^T - s \end{bmatrix},\;\; \boldsymbol{B} = \begin{bmatrix} -\boldsymbol{p} \\ \boldsymbol{p} \end{bmatrix}$$

Finally, the expressions for matrices $\boldsymbol{A}$, $\boldsymbol{X}$ and $\boldsymbol{B}$ can be defined as follows:

$$\boldsymbol{A} = \left(\begin{array}{ccc|c} -x(1) & -y(1) & -z(1) & -1 \\ -x(2) & -y(2) & -z(2) & -1 \\ -x(3) & -y(3) & -z(3) & -1 \\ -x(4) & -y(4) & -z(4) & -1 \\ -x(5) & -y(5) & -z(5) & -1 \\ x(1) & y(1) & z(1) & -1 \\ x(2) & y(2) & z(2) & -1 \\ x(3) & y(3) & z(3) & -1 \\ x(4) & y(4) & z(4) & -1 \\ x(5) & y(5) & z(5) & -1 \end{array}\right), \quad \boldsymbol{X} = \begin{pmatrix} a \\ b \\ c \\ s \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} -p(1) \\ -p(2) \\ -p(3) \\ -p(4) \\ -p(5) \\ p(1) \\ p(2) \\ p(3) \\ p(4) \\ p(5) \end{pmatrix}$$

Furthermore, the $\boldsymbol{f}$ vector can be defined to minimise the auxiliary variable which was introduced to represent the maximum absolute deviation. By minimising the auxiliary variable, the maximum absolute deviation will naturally also be minimised.

$$\boldsymbol{f} = (0 \quad 0 \quad 0 \quad 1)$$

Given the above matrices, linprog can now be used. The function can be called as follows:

```
X = linprog(f, A, B)
```

The resultant $\boldsymbol{X}$ vector will contain the optimal solutions for the variables $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ respectively, in addition to the auxiliary variable. The auxiliary variable result can be ignored since it is just placeholders to simplify the optimisation process. The results from running the linprog function are as follows:

$$\boldsymbol{X} = \begin{pmatrix} a \\ b \\ c \\ s \end{pmatrix} = \begin{pmatrix} 1.1562 \\ -0.1896 \\ -0.6044 \\ 4.1619 \end{pmatrix}, \quad \begin{cases} a = 1.1562 \\ b = -0.1896 \\ c = -0.6044 \end{cases}$$

With the above parameters, the predicted price of material for each observation can now be evaluated as follows:

$$p_{predicted}(n) = 1.1562 \cdot x(n) - 0.1896 \cdot y(n) - 0.6044 \cdot z(n).$$

$$p_{predicted} = \boldsymbol{N} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \cdot \begin{bmatrix} 1.1562 \\ -0.1896 \\ -0.6044 \end{bmatrix} = \begin{bmatrix} 53.8097 \\ 15.8381 \\ 94.1619 \\ 95.8381 \\ 44.1619 \end{bmatrix}$$

1.3. Minimise Sum of Square Deviations

1.3.1. Linear System Equation

To minimise the sum of square deviations, the normal equations approach will be used. This approach requires the linear system to take the form of $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$.

Recall the observation matrices:

$$\boldsymbol{N} = \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix}, \quad \boldsymbol{p} = \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

Substituting in $\boldsymbol{N}$ as $\boldsymbol{A}$ and $\boldsymbol{p}$ as $\boldsymbol{b}$ results in the following expression:

$$\boldsymbol{N} \cdot \boldsymbol{x} = \boldsymbol{p}$$

For this assignment problem, directly substituting in the observation matrix results in an overdetermined system. Therefore, to solve for $\boldsymbol{x}$, the inverse of $\boldsymbol{N}$ must be evaluated. However, a requirement for finding the inverse of a matrix is that such a matrix must be an orthogonal (square) matrix. And because $\boldsymbol{N}$ is overdetermined (not a square matrix), a direct inverse cannot be found. Hence, the pseudo-inverse of $\boldsymbol{N}$ must be computed instead. A linear system of equations can be set up as follows, such that the pseudo-inverse of matrix $\boldsymbol{N}$ can be found.

$$\left(\boldsymbol{N}^{\boldsymbol{T}} \cdot \boldsymbol{N}\right) \cdot \boldsymbol{x} = \boldsymbol{N}^{\boldsymbol{T}} \cdot \boldsymbol{p}$$

There is a final requirement that the first dataset should be emphasised 10 times more than the remaining four datasets. To achieve this, an additional weighting matrix can be introduced. This weighting matrix is a diagonal matrix, equivalent in size to the number of observations. Each element on the diagonal corresponds to the weighting of each observation. To emphasise the first element by 10 requires setting the first element on the diagonal to 10 while keeping the remaining as 1. This is shown as follows:

$$\boldsymbol{W} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore, the final form of the system of linear equations is as follows:

$$\left(\boldsymbol{N}^{\boldsymbol{T}} \cdot \boldsymbol{W} \cdot \boldsymbol{N}\right) \cdot \boldsymbol{x} = \left(\boldsymbol{N}^{\boldsymbol{T}} \cdot \boldsymbol{W}\right) \cdot \boldsymbol{p}$$

$$\begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

1.3.2. Solve using Normal Equations

Given the system of linear equations, the vector $\boldsymbol{x}$ can now be solved as follows:

$$\boldsymbol{x} = \left(\boldsymbol{N^T \cdot W \cdot N}\right)^{-1}\left(\boldsymbol{N^T \cdot W}\right) \cdot \boldsymbol{p}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \left(\begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \right)^{-1} \begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \left(\begin{bmatrix} 900 & 100 & 170 & 169 & 100 \\ 100 & 80 & 30 & 31 & 90 \\ 800 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \right)^{-1} \begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \left(\begin{bmatrix} 158461 & 36339 & 148395 \\ 36339 & 17361 & 36905 \\ 148395 & 36905 & 141325 \end{bmatrix} \right)^{-1} \begin{bmatrix} 90 & 100 & 170 & 169 & 100 \\ 10 & 80 & 30 & 31 & 90 \\ 80 & 140 & 160 & 155 & 90 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.0014 & -0.0177 & -0.0009 & 0.0017 & 0.0139 \\ -0.001 & 0.0001 & -0.0018 & -0.0007 & 0.013 \\ -0.0006 & 0.0196 & 0.0025 & -0.0006 & -0.0173 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.014 & -0.0177 & -0.0009 & 0.0017 & 0.0139 \\ -0.01 & 0.0001 & -0.0018 & -0.0007 & 0.013 \\ -0.006 & 0.0196 & 0.0025 & -0.0006 & -0.0173 \end{bmatrix} \begin{bmatrix} 50 \\ 20 \\ 90 \\ 100 \\ 40 \end{bmatrix} = \begin{bmatrix} 0.9832 \\ -0.2011 \\ -0.44 \end{bmatrix}$$

Therefore:

$$\boldsymbol{x} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0.9832 \\ -0.2011 \\ -0.44 \end{pmatrix}, \quad \begin{cases} a = 0.9832 \\ b = -0.2011 \\ c = -0.44 \end{cases}$$

With the above parameters, the predicted price of material for each observation can now be evaluated as follows:

$$p_{predicted}(n) = 0.9832 \cdot x(n) - 0.2011 \cdot y(n) - 0.44 \cdot z(n).$$

$$p_{predicted} = \boldsymbol{N} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 90 & 10 & 80 \\ 100 & 80 & 140 \\ 170 & 30 & 160 \\ 169 & 31 & 155 \\ 100 & 90 & 90 \end{bmatrix} \cdot \begin{bmatrix} 0.9832 \\ -0.2011 \\ -0.44 \end{bmatrix} = \begin{bmatrix} 51.2781 \\ 20.6354 \\ 90.7135 \\ 91.7293 \\ 40.626 \end{bmatrix}$$

2. Part B – Linear Algebra

2.1. Optimise Signal Magnitude Uniformity on a Sphere

Signal data, represented by b, is collected at 576 sampling points on a spherical surface, as depicted in Figure 1(a). To optimise the uniformity of the signal magnitude on the sphere, specific components are strategically positioned within the 528 meshes on the plane, as shown in Figure 1(b). The optimised signal profile is given by (b' = Ax + b), where b is a vector of the sampled data, x is a vector representing the components placed within the plane's meshes in Figure 1(b), A is a system matrix that describes the contribution of each mesh element on the plane to the signal points on the sphere.

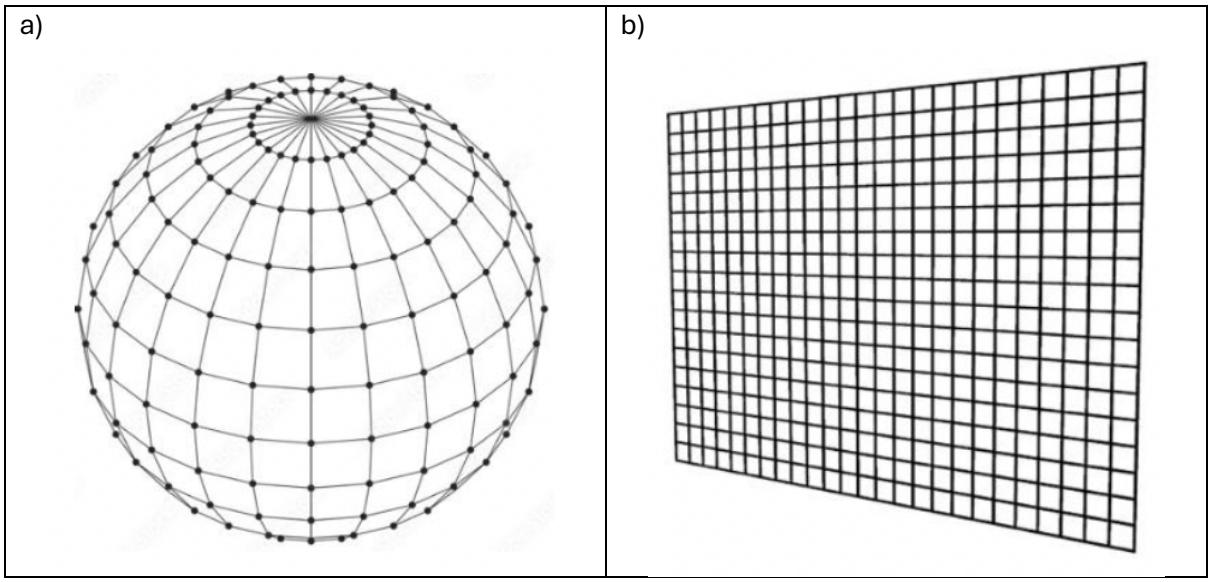

Figure 1 – a) Sampled signal data on sphere. b) Plane to optimise signal data on sphere.

The optimised signal b' will satisfy the following condition:

- **|b' - b0| ≤ ∈ $\mathbf{b0}$, where $\mathbf{b0}$ is the mean value of b'.**
- **∈(m) = 0.0008%, m = 1, 2, ..., M = 576.**

Additionally, both b and b' are positive vectors. For each component in vector x, the following conditions are required:

$$0 \le x(n) + x0(n) \le 0.007,\ n = 1, 2, \ldots, N = 528.$$

where vector x0 has the same dimension as vector x and represents pre-loaded components within the mesh structure depicted in Figure 1(b), indicating that not all meshes are empty before optimisation.

2.1.1. Setting up constraints

Given the parameters provided above, constraints can be defined for both the $\boldsymbol{b}'$ and $\boldsymbol{x}$ vectors. First consider the constraints for $\boldsymbol{b}'$:

$$\boldsymbol{b}' = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}' - \boldsymbol{b}$$

Now consider the following relationship provided in the assignment spec:

$$|\boldsymbol{b}' - b_0| \leq \epsilon \cdot b_0$$

where:

$$b_0 = mean(\boldsymbol{b}')\ ,\ \ \epsilon = 0.000008$$

Additionally, the absolute value expressions on the LHS can be removed by defining a set of two equations that represent the same constraint.

$$\begin{cases} \boldsymbol{b}' - b_0 \leq \epsilon \cdot b_0 \\ b_0 - \boldsymbol{b}' \leq \epsilon \cdot b_0 \end{cases} \tag{2.1.1}$$

The set of equations can be subsequently simplified prior to implementation as shown in the following expressions.

$$\begin{cases} \boldsymbol{b}' \leq \epsilon \cdot b_0 + b_0 \\ -\boldsymbol{b}' \leq \epsilon \cdot b_0 - b_0 \end{cases}$$

$$\begin{cases} \boldsymbol{b}' \leq b_0(\epsilon + 1) \\ -\boldsymbol{b}' \leq b_0(\epsilon - 1) \end{cases}$$

Furthermore, the vector $\boldsymbol{b}$ can be added to the LHS and RHS of both equations as follows. This will allow for a later substitution of the equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}' - \boldsymbol{b}$.

$$\begin{cases} \boldsymbol{b}' - \boldsymbol{b} \leq b_0(\epsilon + 1) - \boldsymbol{b} \\ \boldsymbol{b} - \boldsymbol{b}' \leq b_0(\epsilon - 1) + \boldsymbol{b} \end{cases} \tag{2.1.2}$$

Finally, the expression $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}' - \boldsymbol{b}$ can be substituted into both expressions to obtain the final form of the expression prior to implementation.

$$\begin{cases} \boldsymbol{A}\boldsymbol{x} \leq b_0(\epsilon + 1) - \boldsymbol{b} \\ -\boldsymbol{A}\boldsymbol{x} \leq b_0(\epsilon - 1) + \boldsymbol{b} \end{cases} \tag{2.1.3}$$

Now consider the expression $b_0 = mean(\boldsymbol{b}')$, this can be expanded out into the following form:

$$b_0 = mean(\boldsymbol{b}') = mean(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) = mean(\boldsymbol{A}\boldsymbol{x}) + mean(\boldsymbol{b})$$

Now consider the *mean* of $\boldsymbol{Ax}$ where:

$$mean\left(\boldsymbol{A}_{(m\times n)}\cdot\boldsymbol{x}_{(n\times 1)}\right)=mean\left(\begin{pmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\a_{21}&a_{22}&\cdots&a_{2n}\\\vdots&\vdots&\ddots&\vdots\\a_{m1}&a_{m2}&\cdots&a_{mn}\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}\right)$$

$$=\begin{pmatrix}\frac{a_{11}+a_{21}+\cdots+a_{m1}}{m}&\frac{a_{12}+a_{22}+\cdots+a_{m2}}{m}&\cdots&\frac{a_{1n}+a_{2n}+\cdots+a_{mn}}{m}\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}$$

$$=\frac{1}{m}\cdot\begin{pmatrix}a_{11}+a_{21}+\cdots+a_{m1}&a_{12}+a_{22}+\cdots+a_{m2}&\cdots&a_{1n}+a_{2n}+\cdots+a_{mn}\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}$$

This final expression shows that the mean of $\boldsymbol{Ax}$ can be simply expressed as a vector of constants multiplied by $\boldsymbol{x}$. This vector of constants can be expressed as follows:

$$\boldsymbol{C}=\frac{1}{m}\cdot\begin{pmatrix}a_{11}+a_{21}+\cdots+a_{m1}&a_{12}+a_{22}+\cdots+a_{m2}&\cdots&a_{1n}+a_{2n}+\cdots+a_{mn}\end{pmatrix}$$

With this, the mean of $\boldsymbol{Ax}$ can now be expressed as follows:

$$mean(\boldsymbol{Ax})=\boldsymbol{Cx} \tag{2.1.4}$$

Substituting this into the expression for b_0 gives the following expression:

$$b_0=mean(\boldsymbol{b}')=mean(\boldsymbol{Ax}+\boldsymbol{b})=mean(\boldsymbol{Ax})+mean(\boldsymbol{b})=\boldsymbol{Cx}+mean(\boldsymbol{b})$$

Now the new expression for b0 can be substituted back into the inequality constraints. The updated expressions for these inequality constraints are as follows:

$$\begin{cases}\boldsymbol{Ax}\leq\big(\boldsymbol{Cx}+mean(\boldsymbol{b})\big)(\epsilon+1)-\boldsymbol{b}\\-\boldsymbol{Ax}\leq\big(\boldsymbol{Cx}+mean(\boldsymbol{b})\big)(\epsilon-1)+\boldsymbol{b}\end{cases} \tag{2.1.5}$$

Some rearranging can now be done to ensure that the x-vector is moved to the LHS of each expression, steps are shown as follows:

$$\begin{cases} \boldsymbol{Ax} \leq \boldsymbol{Cx}(\epsilon+1) + mean(\boldsymbol{b})(\epsilon+1) - \boldsymbol{b} \\ -\boldsymbol{Ax} \leq \boldsymbol{Cx}(\epsilon-1) + mean(\boldsymbol{b})(\epsilon-1) + \boldsymbol{b} \end{cases}$$

$$\begin{cases} \boldsymbol{Ax} - \boldsymbol{Cx}(\epsilon+1) \leq mean(\boldsymbol{b})(\epsilon+1) - \boldsymbol{b} \\ -\boldsymbol{Ax} - \boldsymbol{Cx}(\epsilon-1) \leq mean(\boldsymbol{b})(\epsilon-1) + \boldsymbol{b} \end{cases}$$

$$\begin{cases} \boldsymbol{Ax} - \boldsymbol{Cx}(1+\epsilon) \leq mean(\boldsymbol{b})(\epsilon+1) - \boldsymbol{b} \\ \boldsymbol{Cx}(1-\epsilon) - \boldsymbol{Ax} \leq mean(\boldsymbol{b})(\epsilon-1) + \boldsymbol{b} \end{cases}$$

Furthermore, additional constant vectors D1 and D2 can be defined to help simplify the final expression.

$$\begin{cases} \boldsymbol{D}_1 = \boldsymbol{C}(1+\epsilon) \\ \boldsymbol{D}_2 = \boldsymbol{C}(1-\epsilon) \end{cases} \tag{2.1.6}$$

The final inequalities are as follows:

$$\begin{cases} (\boldsymbol{A} - \boldsymbol{D}_1) \cdot \boldsymbol{x} \leq mean(\boldsymbol{b})(\epsilon+1) - \boldsymbol{b} \\ (\boldsymbol{D}_2 - \boldsymbol{A}) \cdot \boldsymbol{x} \leq mean(\boldsymbol{b})(\epsilon-1) + \boldsymbol{b} \end{cases} \tag{2.1.7}$$

In addition to the above inequality constraints, there are also constraints on the upper and lower bounds of x. These are given in the assignment specifications and can be defined as:

$$0 \leq x(n) + x_0(n) \leq 0.007$$

$$-x_0(n) \leq x(n) \leq 0.007 - x_0(n)$$

$$ub = 0.007 - x_0(n)\,, \quad lb = -x_0(n)$$

Given the above definitions for the inequality constraints and bounds, the optimisation problem can now be implemented as shown in the following sections.

2.1.2. Minimise 1-norm of x

Minimising the 1-norm of x requires minimising the sum of absolute deviations. To simplify this problem, an auxiliary vector can be introduced such that:

$$|x(n)| \leq s(n)$$

$$\begin{cases} x(n) \leq s(n) \\ -x(n) \leq s(n) \end{cases}$$

$$\begin{cases} x(n) - s(n) \leq 0 \\ -x(n) - s(n) \leq 0 \end{cases}$$

These absolute value constraints can now be added to the previously defined inequality constraints. The resultant set of constraints are:

$$\begin{cases} (\boldsymbol{A} - \boldsymbol{D}_1) \cdot \boldsymbol{x} \leq mean(\boldsymbol{b})(\epsilon + 1) - \boldsymbol{b} \\ (\boldsymbol{D}_2 - \boldsymbol{A}) \cdot \boldsymbol{x} \leq mean(\boldsymbol{b})(\epsilon - 1) + \boldsymbol{b} \\ \boldsymbol{x} - \boldsymbol{s} \leq 0 \\ -\boldsymbol{x} - \boldsymbol{s} \leq 0 \end{cases}$$

To be used in the MATLAB linprog function, the above equations must be defined as:

$$\boldsymbol{A_{ineq}} \cdot \boldsymbol{x_{ineq}} \leq \boldsymbol{b_{ineq}}$$

where:

$$\boldsymbol{A_{ineq}} = \begin{pmatrix} \boldsymbol{A} - \boldsymbol{D}_1 & \boldsymbol{O} \\ \boldsymbol{D}_2 - \boldsymbol{A} & \boldsymbol{O} \\ \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & -\boldsymbol{I} \end{pmatrix}, \quad \boldsymbol{b_{ineq}} = \begin{pmatrix} mean(\boldsymbol{b})(\epsilon + 1) - \boldsymbol{b} \\ mean(\boldsymbol{b})(\epsilon - 1) + \boldsymbol{b} \\ \boldsymbol{O} \\ \boldsymbol{O} \end{pmatrix}, \quad \boldsymbol{x_{ineq}} = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix}$$

Therefore, the final form of the inequality expression for 1-norm is:

$$\begin{pmatrix} \boldsymbol{A} - \boldsymbol{D}_1 & \boldsymbol{O} \\ \boldsymbol{D}_2 - \boldsymbol{A} & \boldsymbol{O} \\ \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & -\boldsymbol{I} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix} \leq \begin{pmatrix} mean(\boldsymbol{b})(\epsilon + 1) - \boldsymbol{b} \\ mean(\boldsymbol{b})(\epsilon - 1) + \boldsymbol{b} \\ \boldsymbol{O} \\ \boldsymbol{O} \end{pmatrix}$$

Furthermore, the upper and lower bounds must be defined for the vectors **x** and **s**. The bounds for **x** (previously defined) and vector **s** are shown as follows:

$$-\boldsymbol{x_0} \leq \boldsymbol{x} \leq 0.007 - \boldsymbol{x_0}, \quad \boldsymbol{s} \geq 0$$

Therefore, the upper and lower bound vectors are shown as follows:

$$\boldsymbol{lb} = (-\boldsymbol{x_0} \quad \boldsymbol{O}), \quad \boldsymbol{ub} = (0.007 - \boldsymbol{x_o} \quad \infty)$$

Finally, the coefficient vector **f** can be defined as a vector of ones of length **x** plus the length of the auxiliary vector **s**.

The above inequalities can now be combined with the upper and lower bounds into the linprog function as follows:

```
[x, fval] = linprog(f, Aineq, bineq, [], [], lb, ub);
```

The resultant vector **x** following optimisation is shown as follows with a resultant fval of 0.1317.

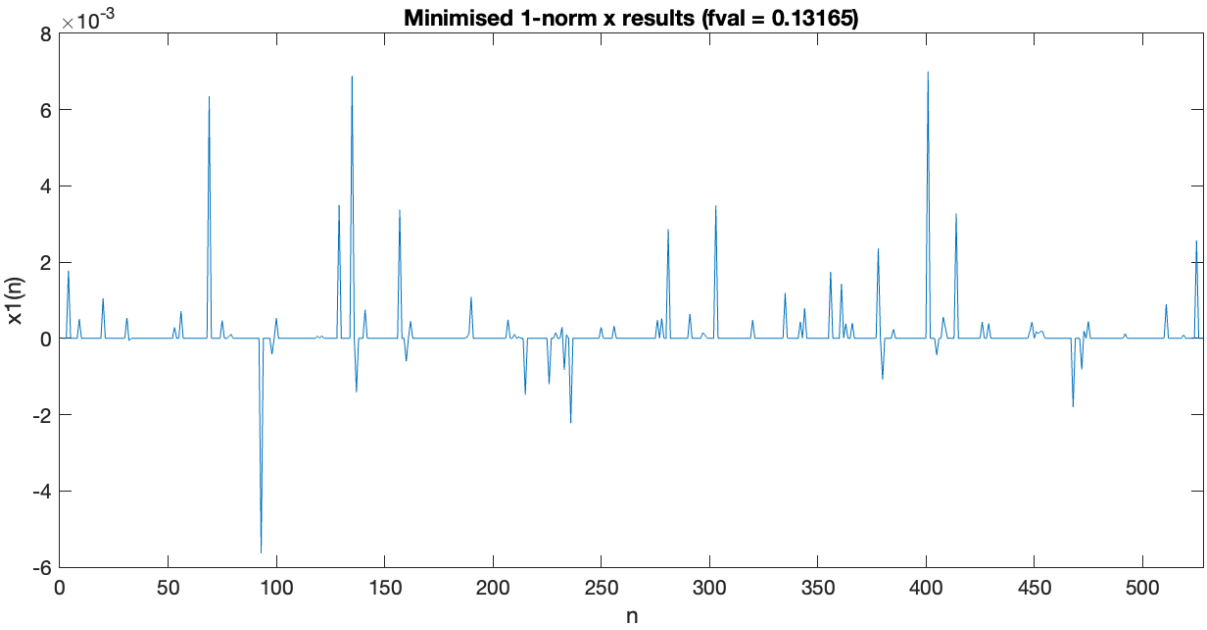

Figure 2.1.1 – Plot of minimised 1-norm x results

As shown, all values of x are quite small in magnitude, between -0.006 and 0.008, with the majority equal to zero which is to be expected.

2.1.3. Minimise 2-norm of x

Minimising the 2-norm of x requires minimizing the square root of the sum of square deviations. The same inequality constraints used for minimising the 1-norm can also be used to minimise the 2-norm, however the absolute value constraints are not required.

$$\begin{cases}(\boldsymbol{A}-\boldsymbol{D}_1)\cdot\boldsymbol{x}\leq mean(\boldsymbol{b})(\epsilon+1)-\boldsymbol{b}\\(\boldsymbol{D}_2-\boldsymbol{A})\cdot\boldsymbol{x}\leq mean(\boldsymbol{b})(\epsilon-1)+\boldsymbol{b}\end{cases}$$

Again, to be used in the MATLAB linprog function, the above equations must be defined as:

$$\boldsymbol{A_{ineq}}\cdot\boldsymbol{x_{ineq}}\leq\boldsymbol{b_{ineq}}$$

where:

$$\boldsymbol{A_{ineq}}=\begin{pmatrix}\boldsymbol{A}-\boldsymbol{D}_1\\\boldsymbol{D}_2-\boldsymbol{A}\end{pmatrix},\quad\boldsymbol{b_{ineq}}=\begin{pmatrix}mean(\boldsymbol{b})(\epsilon+1)-\boldsymbol{b}\\mean(\boldsymbol{b})(\epsilon-1)+\boldsymbol{b}\end{pmatrix},\quad\boldsymbol{x_{ineq}}=(\boldsymbol{x})$$

Therefore, the final form of the inequality expression for 2-norm is:

$$\begin{pmatrix}\boldsymbol{A}-\boldsymbol{D}_1\\\boldsymbol{D}_2-\boldsymbol{A}\end{pmatrix}\cdot(\boldsymbol{x})\leq\begin{pmatrix}mean(\boldsymbol{b})(\epsilon+1)-\boldsymbol{b}\\mean(\boldsymbol{b})(\epsilon-1)+\boldsymbol{b}\end{pmatrix}$$

Furthermore, the upper and lower bounds must be defined for the vector **x**, as previously defined, are:

$$-\boldsymbol{x_0}\leq\boldsymbol{x}\leq0.007-\boldsymbol{x_0}$$

Therefore, the upper and lower bound vectors are shown as follows:

$$\boldsymbol{lb}=(-\boldsymbol{x_0}),\quad\boldsymbol{ub}=(0.007-\boldsymbol{x_o})$$

Finally, the function to minimise (2-norm) can be defined in MATLAB using the norm function as shown:

```
f = @(x) norm(x, 2);
```

The above inequalities can now be combined with the upper and lower bounds into the fmincon function as shown. The default “interior-point” algorithm is used with the maximum number of function evaluations set to 3000 for that algorithm.

```
[x, fval] = fmincon(f, x0, Aineq, bineq, [], [], lb, ub);
```

The resultant vector **x** following optimisation is shown as follows with a resultant fval of 0.00676. Note that the default maximum function evaluations for the fmincon function in MATLAB is set to 3000. It was found that this was not sufficient, therefore the maximum function evaluations value was increased to 100,000 such that a local minimum was found that satisfies the constraints. The results of this are shown as follows.

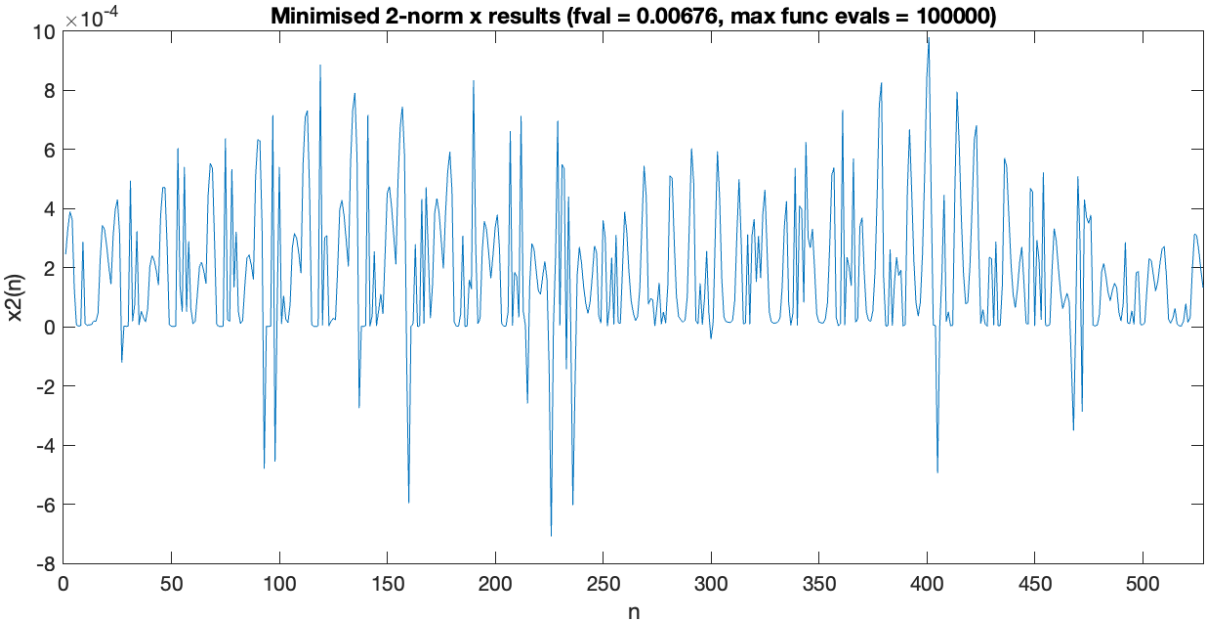

Figure 2.1.2 – Plot of minimised 1-norm x results

As shown, all values of x are quite small in magnitude, between -0.0008 and 0.001. Unlike the result for the 1-norm where most x-values were equal to zero, most of the x-values for the 2-norm result are not equal to zero. That said, the magnitude of these values are still relatively small resulting in a smaller objective function solution (fval) compared to the 1-norm result.

The fmincon function was also run for varying maximum function evaluation values, a summary of which is shown in the following table. The full plots of the results are shown in Appendix A.

Table 2.1.1 – Resultant fval at differing max function eval values

Max Function Evaluations	Obj Function Value (fval) at Solution
3,000	0.01556
10,000	0.01042
30,000	0.00724
100,000	0.00676

2.2. Applications of Singular Value Decomposition (SVD) in Bioinformatics

2.2.1. Introduction to SVD

Singular Value Decomposition (SVD) is a method of decomposing a matrix into three matrices U, S and V that can reveal certain properties about the original matrix to capture essential information regarding the original matrix. Because of this unique property, SVD is widely used in real-world applications such as data compression and analysis. The mathematical definition of SVD is given as follows:

$$\boldsymbol{A = U \cdot \Sigma \cdot V^T}$$

To perform SVD, the following steps are to be performed:

1. Find the left and right singular matrices of A. Let A be an m by n matrix.

$$\boldsymbol{S_L = A \cdot A^T}\ ,\ \ \boldsymbol{S_R = A^T \cdot A}$$

2. Find the left and right singular vectors. These singular vectors are the columns of the left and right singular matrices respectively.

$$\boldsymbol{S_L} = \begin{pmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{u_1} & \boldsymbol{u_2} & \cdots & \boldsymbol{u_m} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix},\ \ \boldsymbol{S_R} = \begin{pmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{v_1} & \boldsymbol{v_2} & \cdots & \boldsymbol{v_n} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix}$$

3. For the eigenvalues of the left and right singular matrices and rank in descending order.

$$\boldsymbol{S_L}:\ \ \lambda_{u_1} > \lambda_{u_2} > \cdots > \lambda_{u_m}$$
$$\boldsymbol{S_R}:\ \ \lambda_{v_1} > \lambda_{v_2} > \cdots > \lambda_{v_n}$$

4. Additionally, the left and right singular matrices are also positive semi-definite (PSD) matrices. This results in both matrices having eigenvalues that are greater than or equals to zero. Furthermore, the left and right singular matrices have identical non-zero eigenvalues. Additionally, assume n > m.

$$\lambda_1 = \lambda_{u_1} = \lambda_{v_1}$$
$$\lambda_2 = \lambda_{u_2} = \lambda_{v_2}$$
$$\cdots$$
$$\lambda_m = \lambda_{u_m} = \lambda_{v_m}$$
$$\lambda_n = 0\ ,\ \ n > m$$

5. The singular values of matrix A can now be found by taking the square roots of the non-zero eigenvalues.

$$\sigma_1 = \sqrt{\lambda_1}$$
$$\sigma_2 = \sqrt{\lambda_2}$$
$$\cdots$$
$$\sigma_m = \sqrt{\lambda_m}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\sigma_1} & 0_{1,2} & \cdots & 0_{1,m} & 0_{1,m+1} & \cdots & 0_{1,n} \\ 0_{2,1} & \boldsymbol{\sigma_2} & \cdots & 0_{2,m} & 0_{2,m+1} & \cdots & 0_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0_{m,1} & 0_{m,2} & \cdots & \boldsymbol{\sigma_m} & 0_{m,m+1} & \cdots & 0_{m,n} \end{pmatrix}$$

6. The U and V matrices can then be constructed using the normalised u and v vectors from the left and right singular matrices. For this, let $\boldsymbol{u}'$ represent the normalised $\boldsymbol{u}$ vector and $\boldsymbol{v}'$ represent the normalised $\boldsymbol{v}$ vector.

$$\boldsymbol{U} = \begin{pmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{u_1'} & \boldsymbol{u_2'} & \cdots & \boldsymbol{u_m'} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix}, \quad \boldsymbol{V} = \begin{pmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{v_1'} & \boldsymbol{v_2'} & \cdots & \boldsymbol{v_n'} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix}$$

7. The final expression is as follows:

$$\boldsymbol{A} = \boldsymbol{U} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{V^T} = \begin{pmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{u_1'} & \boldsymbol{u_2'} & \cdots & \boldsymbol{u_m'} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_m & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \leftarrow & \boldsymbol{v_1'} & \rightarrow \\ \leftarrow & \boldsymbol{v_2'} & \rightarrow \\ \vdots & \vdots & \vdots \\ \leftarrow & \boldsymbol{v_n'} & \rightarrow \end{pmatrix}$$

8. Since the eigenvalues are ranked in descending order in the sigma matrix, the lower eigenvalues can be truncated to approximate the original matrix. This is particularly useful for data compression. However, because truncation is involved, this technique results in lossy data compression.

   Finally, by capturing the matrix into a fewer number of eigenvalues / singular values, data analysis can be more easily performed especially in more complex datasets.

2.2.2. Real-world applications of SVD

Given that SVD is tool where matrices and be decomposed to capture essential features, various real-world applications already exist where SVD is extremely useful. Most of these applications are data compression or analysis based. These include:

1. General dimensionality reduction
2. General data compression
3. Digital signal processing
4. Latent Semantic Analysis (LSA)
5. Principal Component Analysis (PCA)

This report will focus on the use of SVD for data analysis particularly in the genetics and bioinformatics field where SVD can be used to analyse multi-dimensional data such as gene expression profiles, pathways and module identifications etc. These aspects are extremely important in the field of bioinformatics with extensive used in genome sequencing, identifying biomarkers and segmenting or clustering gene data. Given this, the following sections will focus on the theory, results and discussions of specific applications related to bioinformatics.

2.2.3. Pathway level gene expression analysis using SVD

The first paper to be discussed is called *"Pathway level analysis of gene expression using singular value decomposition'* by J. Tomfohr, J. Lu and T. B. Kepler. This paper introduces a method, Pathway Level Analysis of Gene Expression (PLAGE), that analyses genetic data by looking at collections predefined gene sets instead of just individual genes.

The limitation of traditional methods looking at individual genes is the possibility of missing group related behaviours. Individual genes may be found to be statistically insignificant to certain observable characteristics, however these genes may work collectively across larger and more complex groups knows as pathways. These pathways represent a more coordinated set or group of genes that individually may not seem significant to a given biological function but work together to define certain biological characteristics.

To analyse these pathways, PLAGE quantifies the activity of predefined pathways by applying SVD to the gene expression data and extracting the singular values / principal components that represent the collective behaviour of the genes within the pathway. A summary of the implementation is given as follows:

1. Select genes from a list of predefined gene sets / pathways.
2. Construct a data matrix for each gene set where the rows represent the genes and the columns represent the samples.
3. Perform SVD on the given data matrix. The left singular matrix, known as the metagene, can be used to find variations in the data for each sample.
4. The metagene singular values for each sample can then be used in a scoring system for the given pathway. These scores correspond to the behaviour of the collective expression for the pathway in each sample.
5. These pathway expression / activation scores can then be normalised to allow for comparison with other pathways and samples.
6. Finally, statistical thresholding can be applied to the pathway scores to identify statistically important pathways for a given biological characteristic.

Some advantages of the PLAGE method include:

- Simplification of complex data by focussing on the metagene matrix and singular values.
- Less susceptible to noise variations as the analysis takes place on the key components rather than the raw data.
- Ability to quantify genotypical pathway activities and allow direct comparisons to phenotypical observations.

The paper then goes to test PLAGE for two specific pathways. A type II diabetes study and a smoking / airway epithelial cell study was conducted. In both studies, various pathways were concluded to be linked to biological observations, showing the usefulness of the PLAGE method. Finally, some limitations of the PLAGE method include:

- Dependence on pre-defined gene datasets / pathways.
- Assumes linear relationships between genotype and phenotype.
- Requires a large sample size to obtain reliable results when analysing the metagene.

(Tomfohr, Lu, & Kepler, 2005)

2.2.4. Conserved and divergent co-expression gene module identification using SVD

The next paper to be discussed is called *"svdPPCS effective singular value decomposition-based method for conserved and divergent co-expression gene module identification"* by W. Zhang, A. Edwards, W. Fan, D. Zhu and K. Zhang. This paper presents a computational approach to analyse complex gene expression data across various biological categories with the aim of identifying gene modules that are either conserved or divergent. Conserved gene co-expressions occurs when a group of genes maintains similar expression patterns across different biological categories whereas divergent gene co-expressions refer to genes that exhibit different or even opposing expression patterns across biological categories. The ability to conduct these gene expression profiles is an important step in further understanding more complex biological processes and pathways.

The methodology proposed by the paper involves using singular value decomposition with pattern pairing and chart splitting (svdPPCS) to capture essential information about the gene data. The key components of this method are:

- Singular Value Decomposition (SVD)
- Pattern Paring (PP), where patterns found in differing datasets are aligned and compared.
- Chart Splitting (CS), where the expression data is divided into sub-modules based on the pattern pairing results.

The steps for implementing svdPPCS are given as follows:

1. Gene expression data must be collected from different biological categories. This data must first be normalised to ensure comparability.
2. A single combined expression matrix is constructed is defined by combining the aligned genes across multiple categories.
3. The combined expression matrix can then be decomposed into the left and right singular matrices where the left singular matrix represents the expression patterns for each gene while the right singular matrix represents the samples.
4. The conserved and divergent modules can then be identified where conserved modules are identified from singular vectors that capture similar expression patterns across samples whereas divergent modules are identified from singular vectors that show different patterns between categories.
5. Statistical methods are then implemented to determine cutoffs based on statistically significant thresholds.
6. Genes are finally grouped based on their contributions to the singular vectors. This can be ranked by the singular values. The result of this groups the genes into resultant gene co-expression groups.

This method was validated in a simulation study set in controlled conditions where datasets where synthetically generated with known conserved and divergent patterns. The svdPPCS method was then implemented on this dataset and compared with the known patterns. It was shown that the svdPPCS method was not only able to accurately identify the predefined modules but also demonstrated robustness in detecting prominent patterns. However, a limitation of the svdPPCS method is that it often misses subtle divergences due to the nature of SVD assuming linear relationships between the genetic data and the resultant co-expressions.

(Zhang, Edwards, Fan, Zhu, & Zhang, 2010)

2.2.5. Genome-wide expression data processing and modelling using SVD

The third and final paper to be discussed in called *"Singular value decomposition for genome-wide expression data processing and modelling"* by O. Alter, P. O. Brown, and D. Botstein. This paper describes the use of SVD for large-scale data analysis. The paper attempts to address the challenges of working with high-dimensional data by transforming genome-wide datasets into a more condensed and interpretable form.

This paper focusses on the use of SVD for the following aspects:

- Dimensionality reduction
- Identification of dominant patterns
- Noise separation

Starting with the dimensionality reduction aspect, the nature of SVD decomposes matrices into a set of singular values. These values are then arranged in descending order when constructing the sigma matrix and the left and right normalised singular matrices. This reduces the data set size and complexity while preserving significant patterns.

Dominant patters can also be identified through the descending nature of the singular values in the sigma matrix. Dominant patterns usually correspond to larger singular values. Hence, performing SVD on a given matrix simplified the process of extracting the most dominant patterns within a complex dataset.

On the other hand, any noise in the collected data will likely correspond to the smaller singular values in the sigma matrix. These can be easily separated from the remaining data by truncating the sigma matrix such that the smaller singular values are not considered when performing subsequent analysis on the data.

The overall findings of the paper regarding SVD show the following:

- Can be applied to a wide variety of biological datasets.
- Is resilient to missing data and can handle the typical noise of large biological datasets.
- Aids researchers in formulating new hypotheses about a gene function or regulation by revealing more complex underlying patterns.
- Complements other computational and statistical techniques, improving the overall analysis pipeline.

These benefits are quite useful, particularly for biological data where the datasets are usually quite large and multi-dimensional. Additionally, sampling clean biological data is challenging, therefore an analysis method that is resilient to noisy data with missing data points and outliers is also preferred.

However, the limitation of assuming linear relationships may miss any complex non-linear relationships. Because of this, SVD is found to work best within a larger data-processing pipeline / framework.

(Alter, Brown, & Botstein, 2000)

2.2.6. Conclusion

From the analysis of the three papers mentioned above, the benefits of SVD for data analysis is quite evident. The main benefits being:

- Simplification of complex data
- Dimensionality reduction
- Ability to analyse key components
- Resilient to missing / noisy data points
- Efficient to compute

However, the limitations regarding SVD are:

- Linearity assumptions
- Not enough to be used as a stand-alone method

In conclusion, SVD is yet another tool that can be extremely useful, especially when implemented as part of a larger data analysis pipeline. The ability to apply SVD to large datasets without the need for preprocessing makes it a versatile first step in many data analysis pipelines. However, users should be aware of the above-mentioned limitations associated with SVD if used.

3. References

Alter, O., Brown, P. O., & Botstein, D. (2000). *Singular value decomposition for genome-wide expression data processing and modeling*, 1-6.

Tomfohr, J., Lu, J., & Kepler, T. B. (2005). *Pathway level analysis of gene expression using singular value decomposition*, 2-9.

Zhang, W., Edwards, A., Fan, W., Zhu, D., & Zhang, K. (2010). *svdPPCS: an effective singular value decomposition-based method for conserved and divergent co-expression gene module identification*, 2-14.

4. Appendices

4.1. Appendix A – Minimised 2-norm results for differing max func evals value

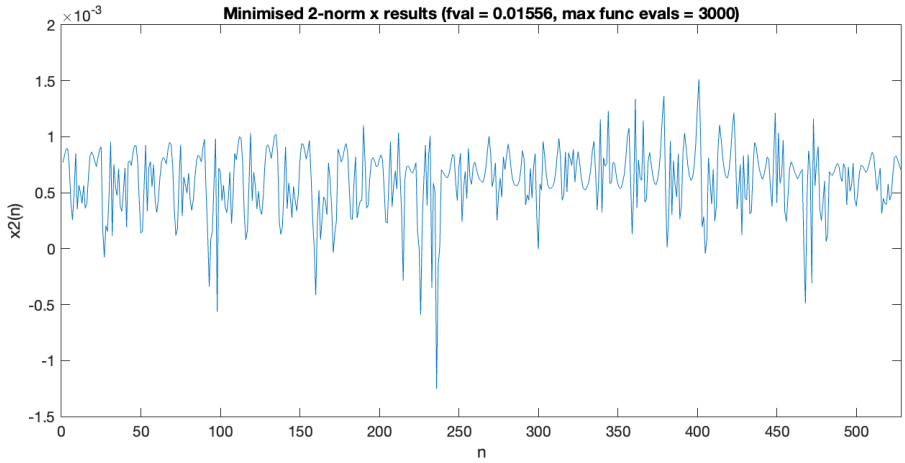

Figure 3.1 – Minimised 2-norm x results for max func evals = 3,000

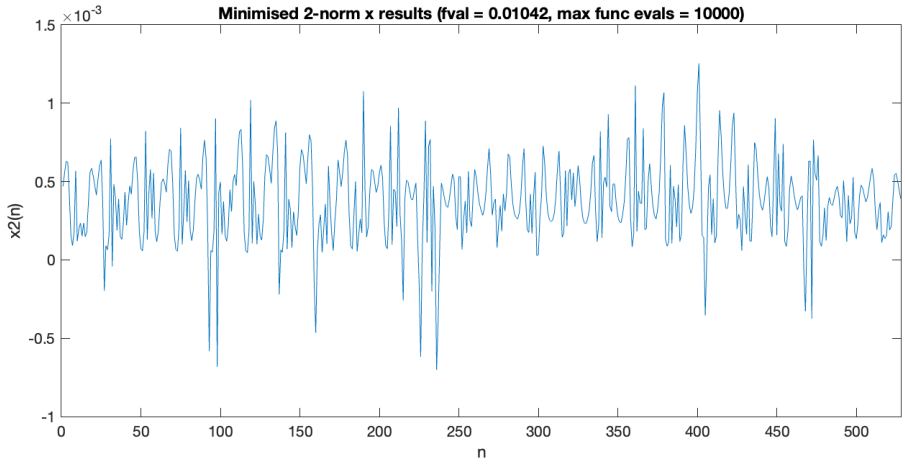

Figure 3.2 – Minimised 2-norm x results for max func evals = 10,000

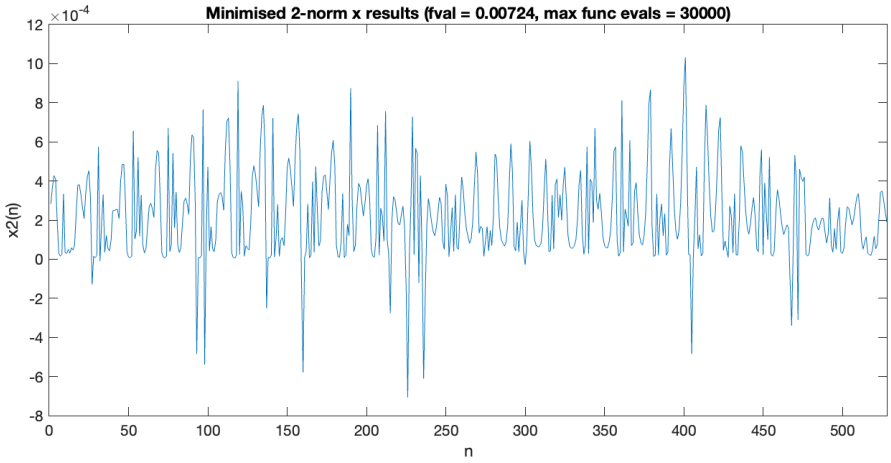

Figure 3.3 – Minimised 2-norm x results for max func evals = 30,000