
DOCUMENTATION

MATLAB CODE FOR
LEARNING TOPOLOGY OF CONSERVED
ARBORESCENCE NETWORKS

WRITTEN BY: SATYA JAYADEV P

PS_JAYADEV@YAHOO.COM

LAST UPDATED ON 19-FEB-2020

Table of Contents

1	Introduction	2
1.1	Network Reconstruction	2
1.2	Types of Nodes	3
1.3	Types of Networks	3
1.4	Conservation Graph	3
2	Methodology	5
2.1	Learning a Model (A)	5
2.1.1	Noisefree Case	5
2.1.2	Noisy Case	6
2.2	Getting f-cutset matrix of Conservation Graph	7
2.3	Graph Realisation from \mathbf{C}_f	7
3	Functionalities of the Code	8
3.1	Generating Random Networks & Flow Data	8
3.2	Inferring Topology from Flow Data	9
4	Functions and Variables	10
4.1	Functions	10
4.1.1	Main	10
4.1.2	Network_Generation	10
4.1.3	Data_Generation	10
4.1.4	Linear_Model	11
4.1.5	Graph_Realisation	11
4.1.6	Graph_Plot	11
4.2	Variables	11
5	Steps to Use the Program	13
5.1	Feed Data \rightarrow Infer Topology	13
5.2	Randomly Generate Network & Data \rightarrow Infer Topology	14
6	Examples	16
6.1	Example 1	16
6.2	Example 2	20
6.3	Example 3	24
7	References	29

Introduction

This document is a comprehensive guide to the MATLAB code to learn the topology of arborescence networks as per the methodology proposed in the paper "Learning Conserved Networks from Data". An arxiv version of this paper is available at [1]. Firstly, the problem of network reconstruction is introduced with some related graph theoretic concepts and then the code is discussed in the later chapters.

1.1 Network Reconstruction

The topology of a complex network with flows along the lines can be represented as a graph. The reconstruction of this underlying graph is referred to as network reconstruction. The topology of a network may be required in various applications depending on the type of the network. In this work, we deal with networks which have certain flow attributes with conservation properties such as power distribution networks, water distribution networks, biological networks, financial networks and so on. The topology of these networks can be represented as directed graphs with the edge directions indicating the direction of flow along that edge. The concepts of spanning trees, cutsets, circuits, incidence matrix, f-cutset matrix, f-circuit matrix, arborescence, spanning arborescence, parent nodes, child nodes and leaf nodes for directed graphs are in consistency with the ones defined in references [2],[3] and [4]. Further, we define certain concepts in relation to the topology of networks with flow conservation properties, which are of interest in this work. These concepts are discussed in the rest of this chapter.

1.2 Types of Nodes

We define three types of nodes which exist in the topology graph of networks of interest:

1. **Source nodes:** The node with only outward oriented edges is called a source node.
2. **Sink Nodes & sink flows:** The node with only inward oriented edges is called a sink node and the flows associated with sink nodes are referred to as sink flows.
3. **Intermediate Nodes:** The node with both inward and outward oriented edges is called an intermediate node.

1.3 Types of Networks

Based on the structure of the topology graph, the networks can be classified as:

1. **Looped Networks:** Networks whose topology graph contains loops or circuits
2. **Non-Looped Networks:** Networks whose topology graph contains loops or circuits
3. **Arborescence Networks:** These are special class of non-looped networks whose topology graph has only single source node

This work deals with network reconstruction of only arborescence networks due to certain special properties they possess (refer to the paper). Arborescence networks may be classified into 3 types of based on the shape and size of the tree structure, as follows:

1. **Binary networks:** Arborescence network in which every parent node has exactly two child nodes
2. **Thin-Long networks:** Arborescence network with many tree layers but fewer nodes at each layer
3. **Fat-Short networks:** Arborescence networks with fewer tree layers but many nodes at each layer.

1.4 Conservation Graph

For certain mathematical convenience, a new graph theoretic concept called conservation graph G_c is defined to be used in the proposed methodology. It is a concept which can be applied to any network topology in general. Conservation graph is one in which conservation equations can be written around every node. In the topology graph, it can be observed that conservation equations cannot be written around source and sink nodes. Therefore, all the source and sink nodes are merged into a single node called the environment node, to get the conservation graph G_c from topology graph. For example, consider an arborescence network with its topology graph and conservation graph shown in Fig. 1.

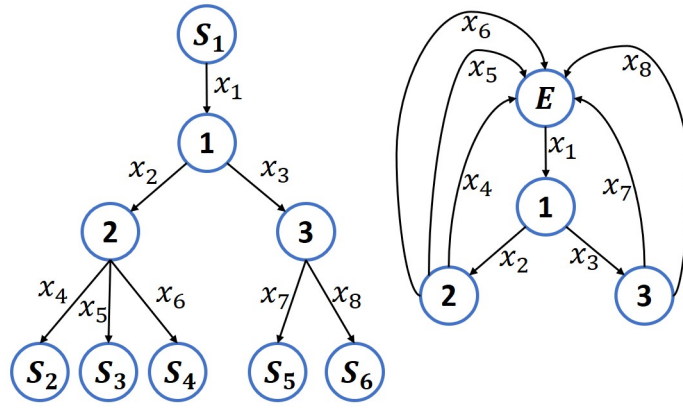


Figure 1: Topology and conservation graphs of an arborescence network

In the conservation graph shown in Fig. reffig:arb, the following conservation equations can be written around every node:

$$\text{At } E: x_1 - x_4 - x_5 - x_6 - x_7 - x_8 = 0$$

$$\text{At } 1: -x_1 + x_2 + x_3 = 0$$

$$\text{At } 2: -x_2 + x_4 + x_5 + x_6 = 0$$

$$\text{At } 3: -x_3 + x_7 + x_8 = 0$$

Methodology

In this chapter, we discuss the methodology presented in the paper, without much explanation. For details about the correctness of the methodology, one may refer the paper. The input to the problem is steady state flow data pertaining to all the edges in the network. Let there be e edges in the network and n_s samples of flows along the edges are collected and stacked into a matrix \mathbf{X} of size $(e \times n_s)$. It is shown that this data is sufficient to recover the topology of an arborescence network.

The following are the major steps in the proposed methodology:

1. Learning a model matrix from data giving relations between the variables (flows) using Singular Value Decomposition (SVD) or Principal Component Analysis (PCA)
2. Transformation of the matrix into an f-cutset matrix of the conservation graph
3. Inferring the topology graph from f-cutset matrix using graph theoretic concepts

2.1 Learning a Model (A)

If the data is noise-free, then SVD can be applied to identify a model describing the relations between variables in the data. If data is noisy, then SVD can be extended to PCA to identify the best fit model. The code is equipped to handle noise-free data as well as noisy data. So we will see how a model can be obtained in both the cases.

2.1.1 Noise-free Case

In this case, when SVD is applied on \mathbf{X} , the last m singular values will be equal to zero where m is the number of linearly independent equations in the model. Therefore, SVD

can be written as:

$$\text{SVD}(\mathbf{X}) = \mathbf{USV} = \mathbf{U}_1 \mathbf{S}_1 \mathbf{V}_1^T + \mathbf{U}_2 \mathbf{S}_2 \mathbf{V}_2^T$$

where \mathbf{U}_1 and \mathbf{V}_1 are the singular vectors corresponding to the non-zero singular values of \mathbf{X} , and \mathbf{U}_2 and \mathbf{V}_2 are the singular vectors corresponding to the *zero* singular values of \mathbf{X} . The model matrix is given by:

$$\mathbf{A} = \mathbf{U}_2^T$$

2.1.2 Noisy Case

Let the data with noisy samples be denoted by \mathbf{Y} . In this case, we assume that the noise in all the variables is uncorrelated. It could be identically distributed i.e., homoscedastic noise or non-identically distributed i.e., heteroscedastic noise. In both these cases, we apply different variants of PCA to identify the best fit model.

1. **Homoscedastic case:** In this case, the smallest m singular values of \mathbf{Y} are not equal to zero but they are small in comparison to other singular values and are equal. Since we are dealing with limited samples, the equality need to be established using hypothesis testing. Therefore, we propose a repetitive hypothesis test to determine the value of m . One may refer to the paper for more details on the hypothesis test. Then again the left singular vectors corresponding to smallest m singular values give the best fit model:

$$\hat{\mathbf{A}} = \mathbf{U}_2^T.$$

2. **Heteroscedastic case:** In this case, the smallest m singular values of \mathbf{Y} are not equal as noise is not identically distributed in all variables. Therefore, \mathbf{Y} is transformed to ensure that the noise in the samples is identically distributed. For this transformation, we assume that the covariance matrix of noise $\mathbf{\Sigma}_e$ is known. In this case also, hypothesis test is applied to determine the value of m after transforming the data matrix. The best fit model is obtained as follows:

$$\begin{aligned}\mathbf{\Sigma}_e &= \mathbf{L}\mathbf{L}^T \\ \mathbf{Y}_s &= \mathbf{L}^{-1}\mathbf{Y} \\ \text{SVD}(\mathbf{Y}_s) &= \mathbf{U}_{1s} \mathbf{S}_{1s} \mathbf{V}_{1s}^T + \mathbf{U}_{2s} \mathbf{S}_{2s} \mathbf{V}_{2s}^T \\ \hat{\mathbf{A}} &= \mathbf{L}^{-1} \mathbf{U}_{2s}^T,\end{aligned}$$

where \mathbf{L} is the Cholesky factor of $\mathbf{\Sigma}_e$.

2.2 Getting f-cutset matrix of Conservation Graph

To determine \mathbf{C}_f of the conservation graph corresponding some spanning tree, we transform the obtained model \mathbf{A} as follows:

$$\begin{aligned}\mathbf{Ax} &= \mathbf{A}_D \mathbf{x}_D + \mathbf{A}_I \mathbf{x}_I = \mathbf{0} \\ \mathbf{x}_D &= -\mathbf{A}_D^{-1} \mathbf{A}_I \mathbf{x}_I \\ \begin{bmatrix} \mathbf{I}_m & \mathbf{A}_D^{-1} \mathbf{A}_I \end{bmatrix} \begin{bmatrix} \mathbf{x}_D \\ \mathbf{x}_I \end{bmatrix} &= \mathbf{C}_f \mathbf{x} = \mathbf{0}\end{aligned}$$

where \mathbf{x}_D and \mathbf{x}_I is a partition of \mathbf{x} such that \mathbf{A}_D is non-singular.

2.3 Graph Realisation from \mathbf{C}_f

Firstly we need to identify the cutset matrix \mathbf{C}_{S_E} corresponding to the spanning arborescence rooted at E , the environment node. Algorithm 1 in the paper is proposed to obtain \mathbf{C}_{S_E} from a \mathbf{C}_f obtained in the previous step. Then, the topology graph of the arborescence network is obtained from \mathbf{C}_{S_E} using Algorithm 2 proposed in the paper. For the algorithms and their correctness, one may refer to the original paper.

Based on this methodology, a MATLAB code has been written to reconstuct the topology of an arborescence network from steady state flow data. The functionalities of the code are detained in the following chapter.

Functionalities of the Code

The MATLAB code is an interactive one in which the user inputs determine what sequence of actions that the program performs. The following are the two major functionalities of the code:

3.1 Generating Random Networks & Flow Data

The code has functions to generate networks randomly based on certain inputs from the user and also generate data which could be either noise-free or noisy. Following are more details about these generators:

1. **Generating a Random Network:** The user can choose to generate a binary network, thin-long network or fat-short network. The user can also choose the number of layers to be present in the network, in a given range. The number of child nodes to each parent node are chosen randomly from a uniform distribution.
2. **Generating Flow Data:** Flow data is generated randomly for all sink flows by drawing samples from normal distributions. Three normal distribution with different means and different variances are chosen and randomly assigned to each of the sink flows. The data for the non-sink flows is determined based on the conservation equations.

The user also has the option to choose to generate noise-free or noisy data. In noisy case, the user can choose to add homoscedastic noise (independent and identically distributed) or heteroscedastic noise (only independent but not identical). For homoscedastic noise, the user has to provide an SNR value based on

which variance of the noise to be added is determine while for heteroscedastic case, the user is expected to provide an array (size e) of SNR values. Accordingly, noise samples are drawn from normal distributions of zero mean and added to the flow samples.

Regarding the number of samples to be generated (n_s), they are generated in multiples of e since e is the minimum number required to apply PCA/SVD to get a model. The users have to choose a multiple z , in a given range, and the problem generates $n_s = ze$ samples.

3.2 Inferring Topology from Flow Data

This functionality of the code is based on the methodology described in Chapter 2. The user can choose between feeding input flow data or using the data generated by the program. The code has the capability to infer topology of the network both with noise free and noisy data, when provided with sufficient number of samples. For a given network, the sufficiency of the samples depends on the SNR value in case noise data. However, the number is fixed in case the data is noise free.

4

SECTION

Functions and Variables

In this chapter, we discuss the different functions and variables that were used in the code along with their details.

4.1 Functions

The following are the different functions defined in the code, each in a different .m file:

4.1.1 Main

This is the main function of the code which is to be run to begin the program. All the user interactions and defaults are coded in this function right from taking input data to generating networks to generating data.

4.1.2 Network_Generation

This function generates a network based on the choice of the user. The code also indexes all the edges randomly. It also generates a vector which represents the network connectivity. The total number of nodes, layers, nodes at each layer and index information of edges are returned by this function.

4.1.3 Data_Generation

This function generates flow data pertaining to the generated network. Depending on the users choice, it also adds Gaussian noise, i.d. or i.i.d., to the dataset. It returns data matrix along with noise covariance matrix to the main function.

4.1.4 Linear_Model

This function identifies a linear model from flow data, either generated or user provided. If the data is noise-free, it checks for zero singular values to find m else if the data is noisy, it performs hypothesis testing to find m , the order of the linear model. It returns the model matrix **A** along with the singular values of data matrix **X**.

4.1.5 Graph_Realisation

This function gets transforms the model matrix **A** into an f-cutset matrix of the conservation graph. The f-cutset matrix is transformed as desired and verified if it has the desired form. If it is in desired form, the vector representation of the network topology is obtained from the desired f-cutset matrix. This function returns the desired f-cutset matrix and vector representation of the network. The vector is compared with the true one to verify if the topology is correctly reconstructed.

4.1.6 Graph_Plot

This function is used to plot the network topology from its vector representation. It uses an inbuilt function in MATLAB.

4.2 Variables

The following are the most important variables defined and used in the code. The ranges and default values of some of these variables, as applicable, are provided in Table 1.

1. **X**: Indicates the dataset(s)
2. **noise_var**: Indicates the noise variance in case data with homoscedastic noise is to be generated
3. **SNR**: Indicates SNR value(s) in case data has heteroscedastic noise
4. **data_flag**: Indicates whether data is fed (1) or generated (2)
5. **noise_flag**: Indicates whether data is noisy (1) or noise-free (0)
6. **network_flag**: Indicates the type of network generated viz. Binary (1), Long-thin (2), Short-fat (3)
7. **noise_case**: Indicates the type of noise to be added to the dataset zero (0) for homoscedastic noise and one (1) for heteroscedastic noise
8. **Nlayers**: Indicates number of layers in the generated network
9. **e**: Indicates the number of edges in the network/graph
10. **l_Nnodes**: Indicates number of nodes at each layer of the generated network
11. **e_index**: Indicates the index of edges at each layer of the generated network

12. **true_index** : Indicates the true connectivity of the generated network in vector form
13. **NSamples** : Indicates multiple by which e is to be multiplied to get the number of samples in the dataset
14. **NSamples** : Indicates number of samples in the dataset
15. **Nrepeats** : Indicates the number of datasets generated
16. **Sigma_e** : Indicates the error covariance matrix of data
17. **Ahat** : Indicates the model explaining the data, given by PCA
18. **sin_vals** : Indicates the singular values of **X**
19. **pred_index** : Indicates the connectivity of the network predicted by the program, in vector form
20. **Cf_desired** : Indicates the desired f-cutset matrix of the arborescence network
21. **avg_time** : Indicates the time taken to find the network; average time in case of multiple datasets
22. **avg_test** : Indicates the success in finding the network; average success in case of multiple datasets

Table 1: Applicable Values & Defaults of Variables

S.No.	Variable	Applicable Values/Range	Default Value
1	data_flag	1,2	1
2	network_flag	1,2,3	1
3	noise_flag	1,0	0
4	noise_case	1,0	0
5	Nmultiple	1 to 100	1
6	Nrepeats	1 to 100	10
7	SNR	5 to 500	50

Steps to Use the Program

In this chapter we discuss the exact steps to be followed by the user while interacting with the program, to achieve a particular objective. As discussed in Chapter 3, the user can use the program for the following objectives:

1. To find the topology of an arborescence network by feeding in flow data.
2. To generate an arborescence network of a particular type and pertaining flow data, and then let the code find its topology from that data.

In both the objectives, the data could be noise-free or noisy with noise characteristics as discussed in Chapter 2.

5.1 Feed Data → Infer Topology

In this objective, the user is expected to provide the following:

1. Data Matrix named \mathbf{X} with exactly e rows and any number of columns
2. In case of noisy data, if it is homoscedastic or heteroscedastic
3. In case of heteroscedastic noise, a column vector of SNR values (size e) corresponding to each flow in the network

The program terminates if any of this information is not provided or provided with incorrect dimensions. The following are the steps to be followed when interacting with the program to achieve this objective:

1. Ensure that **X** and SNR are provided in the workspace
2. Run the main function
3. Choose the option '1 - Feed Data', when asked
4. Program checks for variable '**X**' in the workspace and terminates the program if not available
5. Choose whether the data is 'noisy' or 'noisefree', when asked
6. In noisy case, choose whether the noise is 'homoscedastic' or 'heteroscedastic', when asked
7. In heteroscedastic case, program checks for variable 'SNR' in the workspace and terminates the program if not available
8. Finally, the code runs and gives an output whether the topology could be inferred or not including the time taken

5.2 Randomly Generate Network & Data → Infer Topology

In this objective, the user is has an option to choose the following or let the program run with default values ¹:

1. Type of Network to be generated
2. No. of layers in the network to be generated
3. No. of samples in the dataset to be generated
4. Noisy or noisefree data
5. If noisy, homoscedastic or heteroscedastic noise
6. Noise variance or SNR value to be provided
7. If noisy, choose the number of datasets to be generated for a network

The following are the steps to be followed when interacting with the program to achieve this objective:

1. Run the main function
2. Choose the option '2 - Random network and Data', when asked
3. Choose the type of network to be generated: 1 - Binary, 2 - Long-Thin or 3 - Fat-Short, when asked or Binary network is taken by default
4. Choose the number of layers in the network in the given range or a default value is taken

¹To run the code by default at any step, the user has to just press enter at every step.

5. Choose the number of samples in the dataset as a multiple of e or a default value is taken
6. Choose whether the data is 'noisy' or 'noisefree', when asked or a noisefree case is taken by default
7. In noisy case, choose whether the noise is 'homoscedastic' or 'heteroscedastic', when asked or homoscedastic case is chosen by default
8. In homoscedastic case, choose an noise variance value for noise to be added or a default value is taken
9. In heteroscedastic case, choose an SNR value for noise to be added or a default value is taken
10. In noisy case, choose the number of datasets to be generated for a network or a default value is taken
11. Finally, the code runs and gives an output whether the topology could be inferred or not including the average time taken

6

SECTION

Examples

In this chapter, we discuss some specific examples of running the code to achieve specific objectives.

6.1 Example 1

This example shows how once can feed noisy flow data to the program and get the topology of the network from which the data was collected. We take the arborescence network which was used as an example in the paper. Flow data of 10 samples pertaining to that network, with homoscedastic noise, is fed to the program to see if the network could be reconstructed. The following are the steps taken and a snapshot of the MATLAB command line after each step is shown:

1. **Step 1:** Ensure that the data is provided in the workspace named as variable **X**, having the right dimension

```

Command Window
>> X
X =
    49.8316    48.5960    50.0975    51.0692    50.9538    50.1210    46.7645    57.2208    48.3746    54.7056
    31.3624    30.6317    28.1984    32.0935    28.6851    27.4451    28.2860    32.2142    29.8234    33.6801
     8.2727     9.8552     7.5898    13.2879     8.1792     6.6481     8.9883    12.9731    10.3448    13.1058
    10.8151    10.3852    10.9076    10.4478    10.3481    11.1340    10.4094    10.5476     9.0081    12.6446
    12.0338    10.4224     9.8246     8.4934    10.3428     9.5540     9.1950     8.8460     9.9434     8.0159
    18.4145    17.8977    21.8371    19.0800    22.1513    22.6809    18.4864    25.0732    18.8353    20.9721
     8.1062     9.5551    13.7421    10.5638    10.8595     9.6961     7.1882    12.1337    10.1996    12.7842
    10.3554     8.1805     8.1249     8.3166    11.1476    13.0405    11.0309    13.0139     8.4394     8.0528

fx >>

```

Figure 2: Example 1 - Step 1

2. Step 2: Run the 'Main' function

```

Command Window
>> X
X =
    49.8316    48.5960    50.0975    51.0692    50.9538    50.1210    46.7645    57.2208    48.3746    54.7056
    31.3624    30.6317    28.1984    32.0935    28.6851    27.4451    28.2860    32.2142    29.8234    33.6801
     8.2727     9.8552     7.5898    13.2879     8.1792     6.6481     8.9883    12.9731    10.3448    13.1058
    10.8151    10.3852    10.9076    10.4478    10.3481    11.1340    10.4094    10.5476     9.0081    12.6446
    12.0338    10.4224     9.8246     8.4934    10.3428     9.5540     9.1950     8.8460     9.9434     8.0159
    18.4145    17.8977    21.8371    19.0800    22.1513    22.6809    18.4864    25.0732    18.8353    20.9721
     8.1062     9.5551    13.7421    10.5638    10.8595     9.6961     7.1882    12.1337    10.1996    12.7842
    10.3554     8.1805     8.1249     8.3166    11.1476    13.0405    11.0309    13.0139     8.4394     8.0528

>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated

fx

```

Figure 3: Example 1 - Step 2

- Step 3:** Enter 1 to choose the 'Feed data' option. Programs checks for provision of data and confirms

```

Command Window

>> X

X =

    49.8316    48.5960    50.0975    51.0692    50.9538    50.1210    46.7645    57.2208    48.3746    54.7056
    31.3624    30.6317    28.1984    32.0935    28.6851    27.4451    28.2860    32.2142    29.8234    33.6801
     8.2727     9.8552     7.5898    13.2879     8.1792     6.6481     8.9883    12.9731    10.3448    13.1058
    10.8151    10.3852    10.9076    10.4478    10.3481    11.1340    10.4094    10.5476     9.0081    12.6446
    12.0338    10.4224     9.8246     8.4934    10.3428     9.5540     9.1950     8.8460     9.9434     8.0159
    18.4145    17.8977    21.8371    19.0800    22.1513    22.6809    18.4864    25.0732    18.8353    20.9721
     8.1062     9.5551    13.7421    10.5638    10.8595     9.6961     7.1882    12.1337    10.1996    12.7842
    10.3554     8.1805     8.1249     8.3166    11.1476    13.0405    11.0309    13.0139     8.4394     8.0528

>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
1

Checking the provision of data in the workspace with the variable name X
Data provided

Enter 1 if data is noisy : Enter 0 if data is noisefree
1

```

Figure 4: Example 1 - Step 3

4. Step 4: Enter 1 to inform that the data is noisy

```

Command Window

>> X

X =

    49.8316    48.5960    50.0975    51.0692    50.9538    50.1210    46.7645    57.2208    48.3746    54.7056
    31.3624    30.6317    28.1984    32.0935    28.6851    27.4451    28.2860    32.2142    29.8234    33.6801
     8.2727     9.8552     7.5898    13.2879     8.1792     6.6481     8.9883    12.9731    10.3448    13.1058
    10.8151    10.3852    10.9076    10.4478    10.3481    11.1340    10.4094    10.5476     9.0081    12.6446
    12.0338    10.4224     9.8246     8.4934    10.3428     9.5540     9.1950     8.8460     9.9434     8.0159
    18.4145    17.8977    21.8371    19.0800    22.1513    22.6809    18.4864    25.0732    18.8353    20.9721
     8.1062     9.5551    13.7421    10.5638    10.8595     9.6961     7.1882    12.1337    10.1996    12.7842
    10.3554     8.1805     8.1249     8.3166    11.1476    13.0405    11.0309    13.0139     8.4394     8.0528

>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
1

Checking the provision of data in the workspace with the variable name X
Data provided

Enter 1 if data is noisy : Enter 0 if data is noisefree
1

Enter 1 for homoscedastic case (or) Enter 0 for heteroscedastic case and provide a variable SNR comprising SNR values
1

```

Figure 5: Example 1 - Step 4

5. **Step 5:** Enter 1 to inform that the noise is homoscedastic. Then the rest of the code runs and the user is informed that the network is exactly reconstructed from given data

```

Command Window

>> X

X =

    49.8316    48.5960    50.0975    51.0692    50.9538    50.1210    46.7645    57.2208    48.3746    54.7056
    31.3624    30.6317    28.1984    32.0935    28.6851    27.4451    28.2860    32.2142    29.8234    33.6801
     8.2727     9.8552     7.5898    13.2879     8.1792     6.6481     8.9883    12.9731    10.3448    13.1058
    10.8151    10.3852    10.9076    10.4478    10.3481    11.1340    10.4094    10.5476     9.0081    12.6446
    12.0338    10.4224     9.8246     8.4934    10.3428     9.5540     9.1950     8.8460     9.9434     8.0159
    18.4145    17.8977    21.8371    19.0800    22.1513    22.6809    18.4864    25.0732    18.8353    20.9721
     8.1062     9.5551    13.7421    10.5638    10.8595     9.6961     7.1882    12.1337    10.1996    12.7842
    10.3554     8.1805     8.1249     8.3166    11.1476    13.0405    11.0309    13.0139     8.4394     8.0528

>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
1

Checking the provision of data in the workspace with the variable name X
Data provided

Enter 1 if data is noisy : Enter 0 if data is noise free
1

Enter 1 for homoscedastic case (or) Enter 0 for heteroscedastic case and provide a variable SNR comprising SNR values
1
Homoscedastic Case
The arborescence network is exactly identified from given data in 0.73 seconds and plotted as a graph (tree)
fx >> |

```

Figure 6: Example 1 - Step 5

6. **Step 6:** Check the figure generated by the code. The directed graph matches the topology of the network pertaining to which the data was generated.

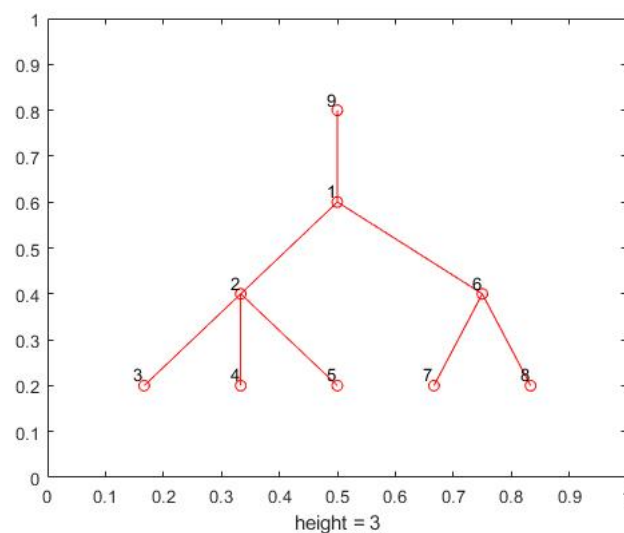


Figure 7: Example 1 - Step 6

6.2 Example 2

This example shows how once can generate a binary network of 5 layers and noise free data with $2e$ number of samples pertaining to that network and then let the program identify topology from the generated data. The following are the steps taken and a snapshot of the MATLAB command line after each step is shown:

1. **Step 1:** Run the 'Main' function

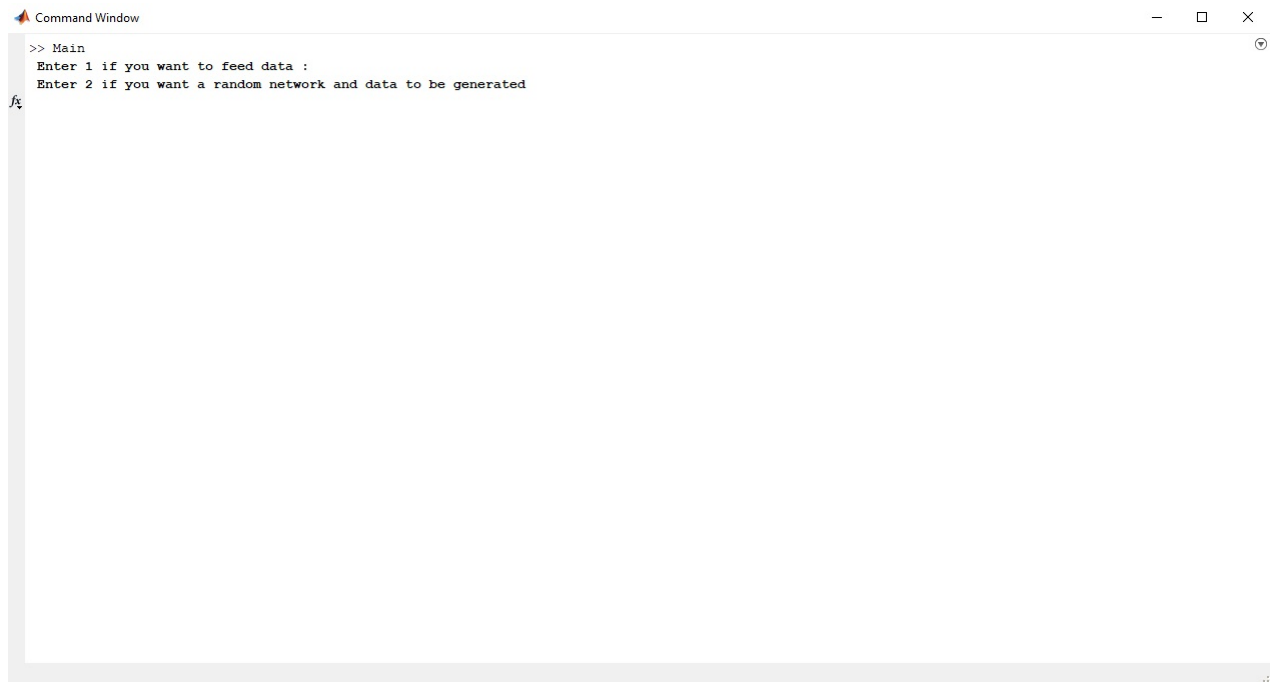


Figure 8: Example 2 - Step 1

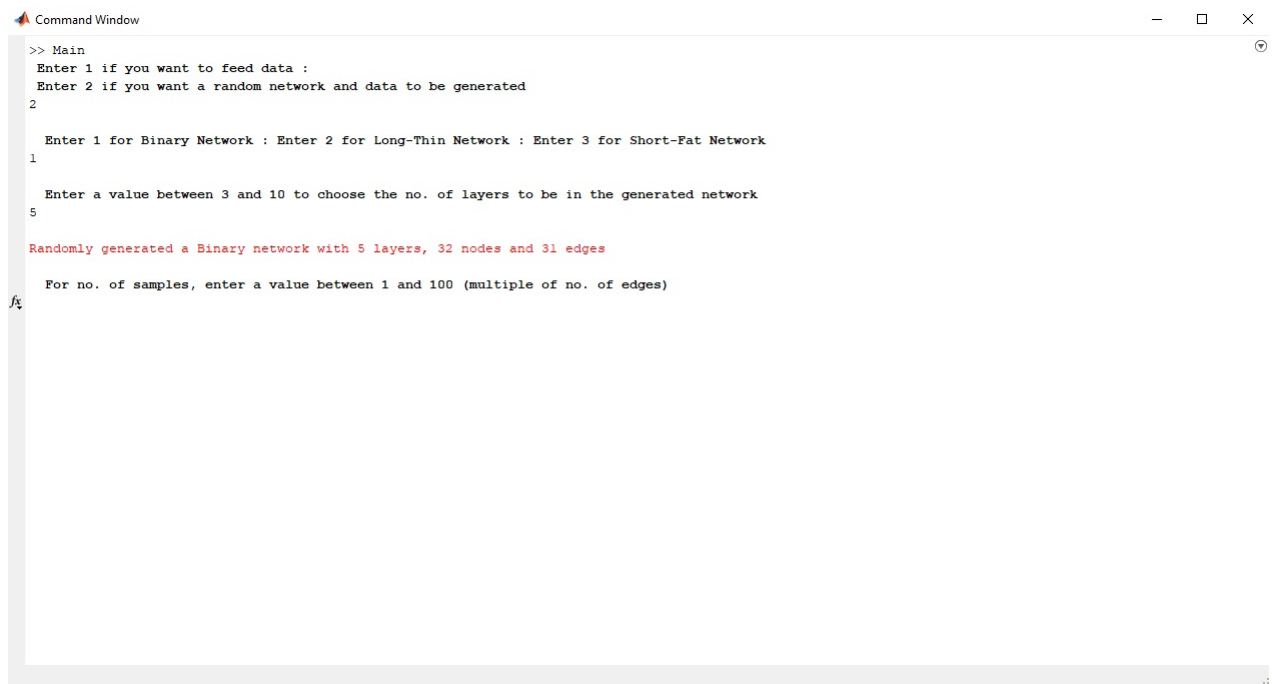
2. **Step 2:** Enter 2 to choose the 'Generate network and data' option.

Figure 9: Example 2 - Step 2

3. **Step 3:** Enter 1 to choose the 'Binary network' option

Figure 10: Example 2 - Step 3

4. **Step 4:** Enter 5 to inform that the network should have 5 layers. The program informs that a binary network has been generated.



```
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
1

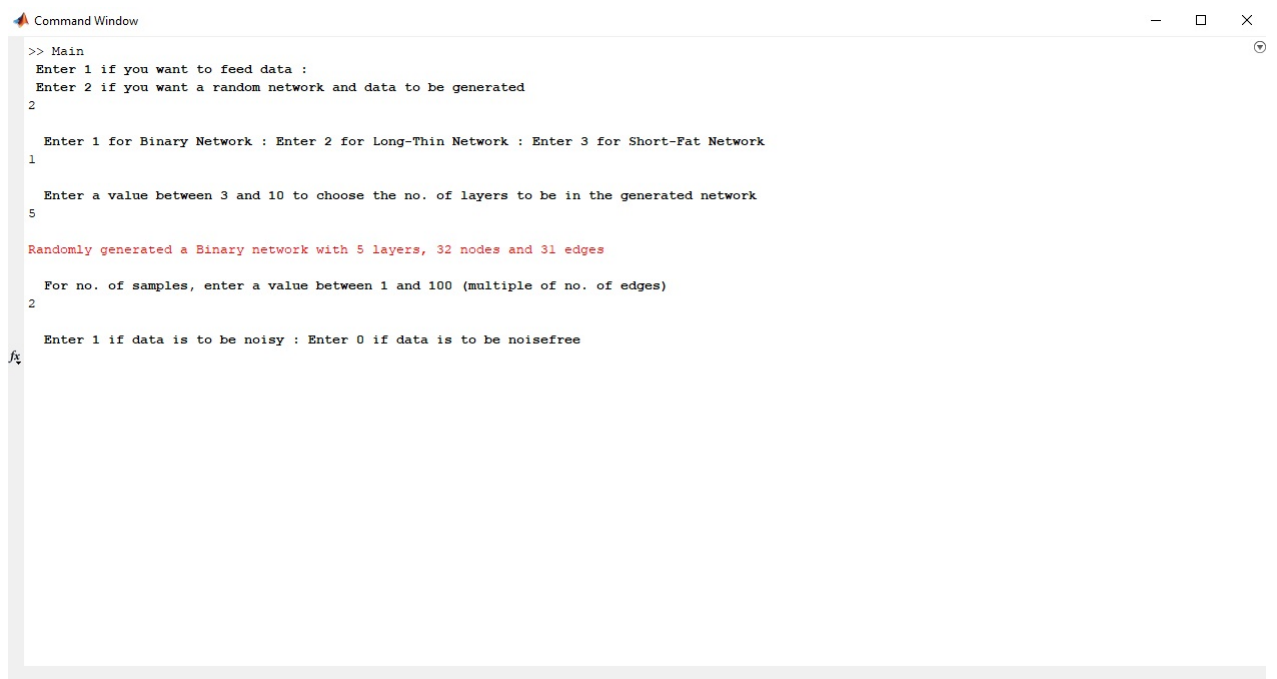
Enter a value between 3 and 10 to choose the no. of layers to be in the generated network
5

Randomly generated a Binary network with 5 layers, 32 nodes and 31 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
```

Figure 11: Example 2 - Step 4

5. **Step 5:** Enter 2 to inform the code to generate $2e$ samples



```
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
1

Enter a value between 3 and 10 to choose the no. of layers to be in the generated network
5

Randomly generated a Binary network with 5 layers, 32 nodes and 31 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
2

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
```

Figure 12: Example 2 - Step 5

6. **Step 6:** Enter 0 to inform that the samples should be noise-free. The program generates a dataset with 62 samples. Then the program uses this data to identify the topology and informs that the network identified matches the one which was generated.

```

Command Window
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
1

Enter a value between 3 and 10 to choose the no. of layers to be in the generated network
5

Randomly generated a Binary network with 5 layers, 32 nodes and 31 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
2

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
0

Randomly generating a noisefree dataset with 62 samples
The arborescence network is exactly identified from given data in 0.16 seconds and plotted as a graph (tree)
fx >>

```

Figure 13: Example 2 - Step 6

- Step 7:** Check the directed graph generated by the code. The height in the figure actually indicates the number of layers in the network.

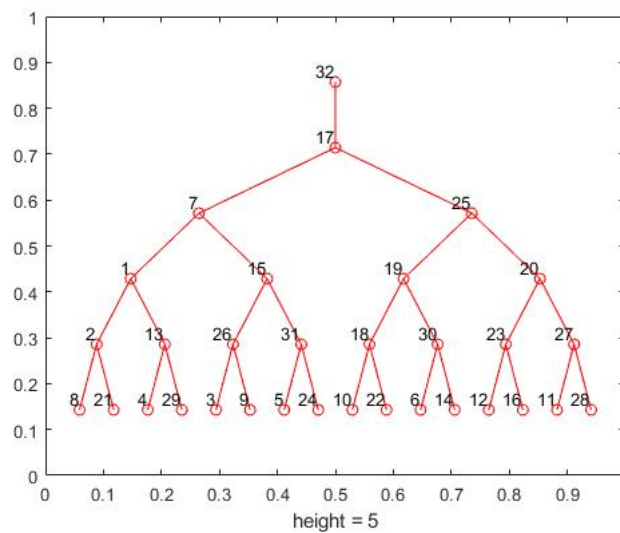
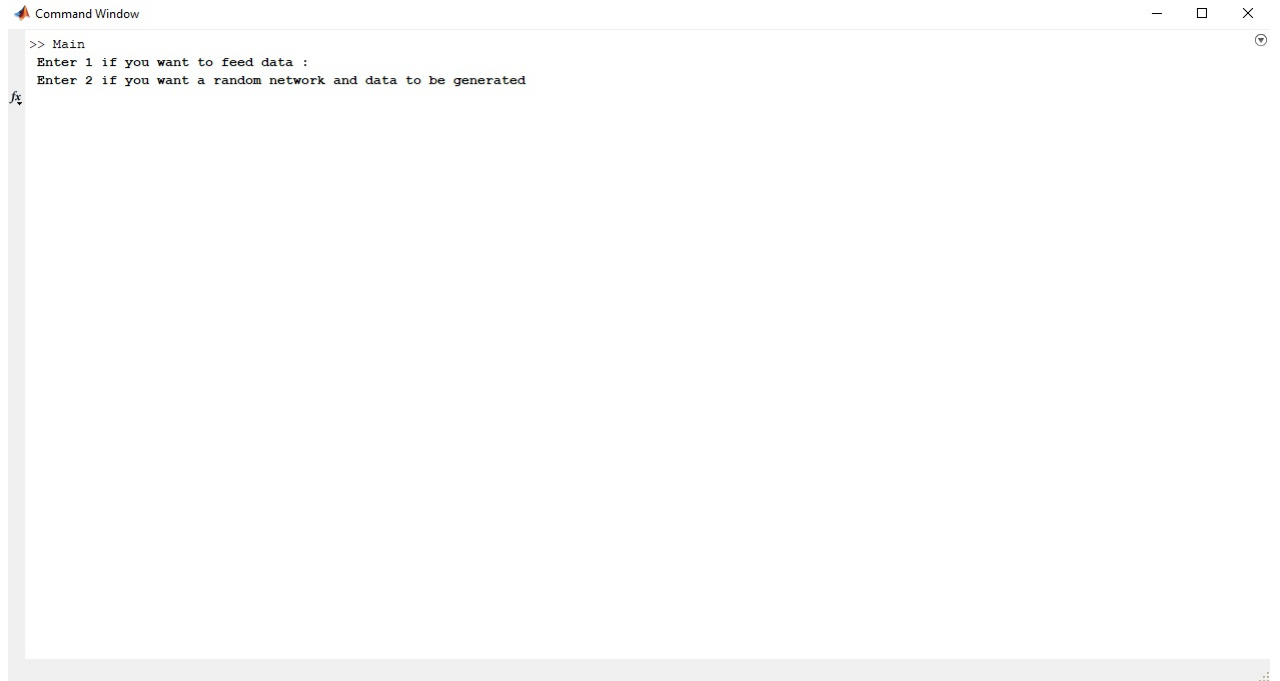


Figure 14: Example 2 - Step 7

6.3 Example 3

This example shows how once can generate a long-thin network of 4 layers. The code is asked to generate 10 datasets each of $3e$ samples with homoscedastic noise of variance 5 pertaining to the network. Then the program goes on to identify topology from each of the generated datasets. The following are the steps taken and a snapshot of the MATLAB command line after each step is shown:

1. **Step 1:** Run the 'Main' function



```
Command Window
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
```

Figure 15: Example 3 - Step 1

2. **Step 2:** Enter 2 to choose the 'Generate network and data' option.



```
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
```

Figure 16: Example 3 - Step 2

3. **Step 3:** Enter 3 to choose the ‘Short-Fat network’ option



```
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network
```

Figure 17: Example 3 - Step 3

4. **Step 4:** Enter 4 to inform that the network should have 4 layers. The program informs that a short-fat network has been generated.

```

Command Window
New to MATLAB? See resources for Getting Started
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network
4

Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
fx

```

Figure 18: Example 3 - Step 4

5. **Step 5:** Enter 3 to inform the code to generate 3e samples

```

Command Window
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network
4

Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
3

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
fx

```

Figure 19: Example 3 - Step 5

6. **Step 6:** Enter 1 to inform that the samples should be noisy

```

Command Window
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network
4

Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
3

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
1

Enter 0 for homoscedastic case (or) Enter 1 for heteroscedastic case
fx

```

Figure 20: Example 3 - Step 6

7. **Step 7:** Enter 0 to inform that the noise should be homoscedastic in nature

```

Command Window
>> Main
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated
2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network
4

Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
3

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
1

Enter 0 for homoscedastic case (or) Enter 1 for heteroscedastic case
0

Choose noise variance in the range of 1-20 for homoscedastic case

```

Figure 21: Example 3 - Step 7

8. **Step 8:** Enter 5 to inform that homoscedastic noise of variance 5 is to be added to the samples

```

Command Window
Enter 1 if you want to feed data :
Enter 2 if you want a random network and data to be generated

2

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network

3

Enter a value between 3 and 5 to choose the number of layers to be in the generated network

4

Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges

For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)

3

Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree

1

Enter 0 for homoscedastic case (or) Enter 1 for heteroscedastic case

0

Choose noise variance in the range of 1-20 for homoscedastic case

5

Enter a value between 1-100 for the no. of datasets to be generated with different noise samples,
fx|
<

```

Figure 22: Example 3 - Step 8

9. **Step 9:** Enter 10 to inform that that 10 datasets are to be generated. The program generates 10 datasets each with 558 samples each. Then the program tries to infer the topology from each of the datasets indepedently. It concludes that the network could not be reconstructed correctly due to insufficient samples for the chosen noise level.

```
Command Window

Enter 1 for Binary Network : Enter 2 for Long-Thin Network : Enter 3 for Short-Fat Network
3
Enter a value between 3 and 5 to choose the number of layers to be in the generated network
4
Randomly generated a Short-Fat network with 4 layers, 187 nodes and 186 edges
For no. of samples, enter a value between 1 and 100 (multiple of no. of edges)
3
Enter 1 if data is to be noisy : Enter 0 if data is to be noisefree
1
Enter 0 for homoscedastic case (or) Enter 1 for heteroscedastic case
0
Choose noise variance in the range of 1-20 for homoscedastic case
5
Enter a value between 1-100 for the no. of datasets to be generated with different noise samples,
10
Randomly generating 10 noisy datasets with 558 samples each
The arborescence network could not be identified from any of 10 dataset(s) that were generated
fx>>
<
```

Figure 23: Example 3 - Step 9

References

- [1] Satya Jayadev, P. and Narasimhan, S. and Bhatt, N., 2019. Learning Conserved Networks from Flow. arxiv, 1905.08716.
- [2] Bondy, J.A. and Murty, U.S.R., 1976. Graph theory with applications (Vol. 290). London: Macmillan.
- [3] Deo, N., 2017. Graph theory with applications to engineering and computer science. Courier Dover Publications.
- [4] Chen, W.K., 1997. Graph theory and its engineering applications (Vol. 5). World Scientific Publishing Company.
- [5] Fujishige, S., 1980. An efficient PQ-graph algorithm for solving the graph-realization problem. Journal of Computer and System Sciences, 21(1), pp.63-86.
- [6] Bixby, R.E. and Wagner, D.K., 1988. An almost linear-time algorithm for graph realization. Mathematics of operations research, 13(1), pp.99-123.
- [7] Whitney, H., 1933. 2-isomorphic graphs. American Journal of Mathematics, 55(1), pp.245-254.
- [8] Narasimhan, S. and Bhatt, N., 2015. Deconstructing principal component analysis using a data reconciliation perspective. Computers & Chemical Engineering, 77, pp.74-84.
- [9] Jolliffe, I.T., 1986. Principal components in regression analysis. In Principal component analysis (pp. 129-155). Springer, New York, NY.