DOCUMENTATION

MATLAB CODE FOR DIRECTED LINK PREDICTION

WRITTEN BY: SATYA JAYADEV P

PS_JAYADEV@YAHOO.COM

LAST UPDATED ON 25-NOV-2020

Table of Contents

SECTION 1

Introduction

This document is a comprehensive guide to the MATLAB code to predict the links and directionality of flows associated with them in conserved networks as per the methodology proposed in the paper **"Predicting Unknown Directed Links of Conserved Networks from Flow Data"** authored by **Satya Jayadev P**, **Ramkrishna Pasumarthy** and **Nirav Bhatt**. The code requires **MATLAB version R2019a** and higher for all the features to be functional. Firstly, the problem of link prediction is introduced with some related graph theoretic concepts and then the code is discussed in the later chapters.

1.1 Learning Network Topology

The topology of a complex network with flows along the lines can be represented as a graph. Finding the unknown edge connectivities in this graph from data is referred to as link prediction. The graph 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 consistence 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 chap-

ter.

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 contain loops or circuits
2. **Non-Looped Networks:** Networks whose topology graph do not contain loops or circuits
3. **Arborescence Networks:** These are special class of non-looped networks whose topology graph has only single source node

In our previous work [1], we showed that it is possible to recover the topology of an arborescence network from flow data. This work focuses on predicting the unknown links of both looped and non-looped networks which are having non-arborescence structure. The methodology proposed in the paper is discussed in the next chapter.

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 conserved network in general. Conservation graph is a topological representation of the network in which conservation equations can be written around every node. In a conserved network, it can be observed that conservation equations cannot be written around source nodes 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 of the network with n_c nodes and e edges.
For example, consider a network with its topology and conservation graph shown in Fig. 1.

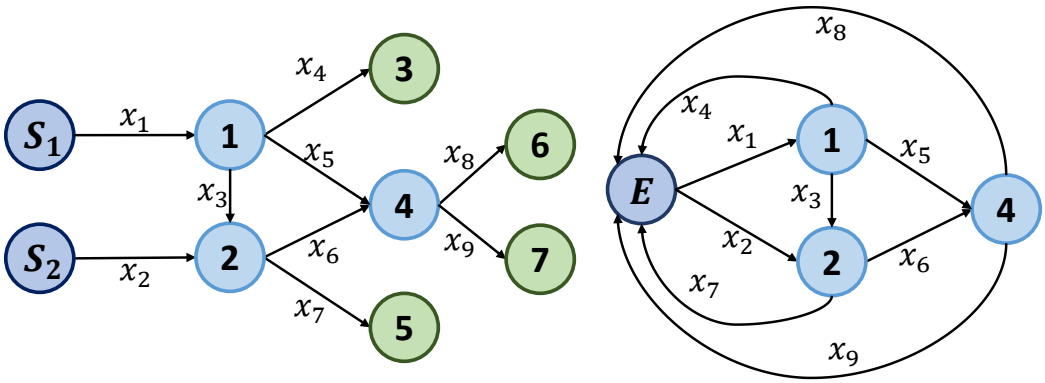

Figure 1: Topology and Conservation graph of a network

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

At E: $x_1 + x_2 - x_4 - x_7 - x_8 - x_9 = 0$

At 1: $-x_1 + x_3 + x_5 + x_4 = 0$

At 2: $-x_2 - x_3 + x_6 + x_7 = 0$

At 4: $-x_5 - x_6 + x_8 + x_9 = 0$

2

SECTION

Methodology

In this chapter, we discuss the methodology presented in the paper, with little 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 and incidence information of known links. 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 $(e \times n_s)$. Let the incidence data of known links be given in the form of a matrix $\mathbf{A}_g$ with each column of $\mathbf{A}_g$ corresponding to a link (or edge). It is shown that $\mathbf{X}$ and $\mathbf{A}_g$ are sufficient to predict the connectivity of all the unknown links in the conservation graph of the network provided $\mathbf{A}_g$ has rank of atleast $n_c - 1$.
The following the major steps in the proposed methodology:

1. Getting the incidence information of a spanning tree, $\mathbf{A}_b$ from $\mathbf{A}_g$, if rank condition is satisfied
2. Learning a model matrix $\mathbf{A}$ from data giving relations between the variables (flows) using Singular Value Decomposition (SVD) or Principal Component Analysis (PCA)
3. Transformation $\mathbf{A}$ into an f-cutset matrix $\mathbf{C}_f$ of the conservation graph corresponding to the spanning tree formed by edges corresponding to columns of $\mathbf{A}_b$
4. Finding the incidence matrix of conservation graph from $\mathbf{C}_f$ and $\mathbf{A}_b$

2.1 Getting $\mathbf{A}_b$ from $\mathbf{A}_g$

$\mathbf{A}_b$ is the matrix which contains just incidence information of a spanning tree of the conservation graph required to get its complete incidence matrix. $\mathbf{A}_b$ is a square matrix

with m rows and columns and is of full rank. Given $\mathbf{A}_g$, to obtain $\mathbf{A}_b$, take the following steps:

1. If $\text{rank}(\mathbf{A}_g) < m$, conclude that the connectivity information is not sufficient to find the incidence matrix.

2. If $\text{rank}(\mathbf{A}_g) \geq m$, then select m linearly independent rows and columns of $\mathbf{A}_g$ to form $\mathbf{A}_b$.

2.2 Learning a Model ($\mathbf{A}$)

If the data is noisefree, 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 noisefree data as well as noisy data. So we will see how a model can be obtained in both the cases.

2.2.1 Noisefree 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:

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

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:

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

2.2.2 Noisy Case

Let the data with noisy samples be denoted by $\mathbf{Y}$. There are two noise cases considered here. Homoscedastic noise in which the noise is i.i.d (independent and identically distributed). Heteroscedastic noise in which the noise is not identically distribution and could even be correlated but the noise covariance matrix is assumed to be known. 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 references [1,8] for more details on handling noisy test and the proposed 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:

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

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

2.3 Getting f-cutset matrix of Conservation Graph

Now, we determine $\mathbf{C}_f$ of the conservation graph corresponding the spanning tree formed by edge corresponding to columns of $\mathbf{A}_b$. If the edges forming $\mathbf{A}_b$ are given by $\mathbf{x}_d$ and rest by $\mathbf{x}_i$, we get $\mathbf{C}_f$ by the following transformation:

$$\begin{aligned}\mathbf{A}\mathbf{x} &= \mathbf{A}_d\mathbf{x}_d + \mathbf{A}_i\mathbf{x}_i = \mathbf{0} \\ \mathbf{x}_d &= -\mathbf{A}_d^{-1}\mathbf{A}_i\mathbf{x}_i \\ \begin{bmatrix}\mathbf{I}_m & \mathbf{A}_d^{-1}\mathbf{A}_d\end{bmatrix}\begin{bmatrix}\mathbf{x}_d \\ \mathbf{x}_i\end{bmatrix} &= \mathbf{C}_f\mathbf{x} = \mathbf{0}.\end{aligned}$$

2.4 Finding Incidence matrix of G_c

Firstly we find a reduced incidence matrix using the following formula derived in the paper:

$$\mathbf{A}_r = \mathbf{A}_b\mathbf{c}_f \tag{1}$$

Finally, the incidence matrix $\mathbf{A}_{in}$ of G_c is obtained from $\mathbf{A}_r$ by adding a row with ± 1 elements in columns having only one non-zero element. The sign should be opposite to that of the non-zero element in each column since we are dealing with a directed graph.

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

SECTION 3

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 noisefree or noisy. Following are more details about these generators:

1. **Generating a Random Network:** The user can choose to generate a random network based on one of the three network models viz. Erdos-Renyi networks [10], Watts–Strogatz networks [11] or Barabasi–Albert networks [12]. The user also has the option to choose the number of nodes in the network. The parameters of the network models are fixed in the code assuming the user might not be familiar with all these models. However, users familiar with these models may alter the parameters in the source code. While converting the topology graph to conservation graph, three could be edges forming self loops at the environment. These are edges with terminal nodes being source and sink nodes. Any such edges generated by the models are removed while constructing the conservation graph of the network.

2. **Generating Flow Data:** Flow data is generated randomly by choosing certain flows to be independent and drawing samples from normal distributions. Three nor-

mal distribution with different means and different variances are chosen and randomly assigned to each of the independent flows. The data for the rest of the flows (dependent ones) is determined based on the nodal conservation equations.

The user also has the option to choose to generate noisefree or noisy data. In noisy case, the user an 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 Finding Incidence Matrix from Flow Data & connectivity of known links

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 network & data generated by the program. The code has the capability to finding the incidence matrix of the G_c both with noisefree and noisy data, when provided with sufficient number of samples and partial incidence information.

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. It also comprises the code which finds the incidence matrix of the graph using the formula after obtaining a linear model.

4.1.2 Network_Generation

This function generates a network based on the choice of the user. The user also has a choice to choose the number of nodes in the network generated. It converts the network into a conservation graph and any edges which form self loops are the environment node are removed. Since there is merger of nodes, the number of nodes in the conservation graph is less than the nodes in the network generated. Then all the edges in the conservation graph are randomly indexed. The code also identifies a spanning tree of the conservation graph. It returns the incidence matrix of G_c, incidence of identified spanning tree along with branch & chord information and number of nodes and edges in G_c.

4.1.3 erdosRenyi

This function is used to generate an Erdos-Renyi network given the number of nodes and other parameters based on the Erdos Renyi model. This is function taken from the MATLAB file exchange [13].

4.1.4 smallw

This function is used to generate an Watts–Strogatz network given the number of nodes and other parameters based on the Watts-Strogatz model. This is a function provided by Mathworks as an example.

4.1.5 BAgraph_dir

This function is used to generate an Barabasi-Albert network given the number of nodes and other parameters based on the Barabasi-Albert model. This is function taken from the MATLAB file exchange [14].

4.1.6 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.

4.1.7 Linear_Model

This function identifies a linear model from flow data, either generated or user provided. If the data is noisefree, 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**.

4.1.8 Plot_Graph

This function plots the conservation graph of a network given the incidence matrix as argument. This is not called from the main function because it has a drawback. Since MATLAB does not allow plotting of parallel edges in directed graphs, any parallel edges in the conservation graph are merged into one edge while plotting the graph. Due to drawback, this function is only provided as a function which user needs to call separately by giving the Incidence matrix as input.

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) provided or generated
2. **SNR :** Indicates SNR value(s) in case data is noisy

3. **data_flag :** Indicates whether data is fed (1) or generated (2)
4. **noise_flag :** Indicates whether data is noisy (1) or noisefree (0)
5. **network_flag :** Indicates the type of network generated viz. Erdos-Renyi (1), Watts–Strogatz (2), Scale-Free (3)
6. **n :** Indicates the user choice of number of the nodes to be present in the network
7. **Adj :** Indicates the adjacency matrix of the undirected network graph generated by the chosen network model
8. **Adj_dir :** Indicates the adjacency matrix of the network graph generated by the chosen network model after assigning directions randomly
9. **Inc_dir :** Indicates the incidence matrix of the network graph generated by the chosen network model after assigning directions randomly
10. **Inc_Con :** Indicates the incidence matrix of G_c after merging the source and sink nodes
11. **Adj_Con :** Indicates the adjacency matrix of G_c
12. **e_c :** Indicates the number of edges in G_c
13. **n_c :** Indicates the number of nodes in G_c
14. **Inc_Con :** Indicates the incidence matrix of the G_c of generated network
15. **Cc_Con :** Indicates the f-cutset matrix pertaining to a spanning tree of G_c identified by the program
16. **Ab :** Indicates the incidence sub-matrix provided or pertaining to the identified spanning tree
17. **Ab_edges :** Indicates the edges corresponding to the columns of incidence sub-matrix provided
18. **branch :** Indicates the edges corresponding to the branches of the identified spanning tree
19. **chord :** Indicates the edges corresponding to the chords/links of the identified spanning tree
20. **b :** Indicates the number of branches in the identified spanning tree
21. **c :** Indicates the number of chords in the identified spanning tree
22. **NSamples :** Indicates multiple by which e is to be multiplied to get the number of samples in the dataset
23. **NSamples :** Indicates number of samples in the dataset

24. **Nrepeats :** Indicates the number of datasets generated

25. **Sigma_e :** Indicates the error covariance matrix of data

26. **Ahat :** Indicates the model explaning the data, given by PCA

27. **sin_vals :** Indicates the singular values of **X**

28. **pred_index :** Indicates the connectivity of the network predicted by the program, in vector form

29. **Cf_desired :** Indicates the desired f-cutset matrix of the arborescence network

30. **avg_time :** Indicates the time taken to find the network; average time in case of multiple datasets

31. **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	n	10 to 100	10
5	Nmultiple	1 to 100	1
6	Nrepeats	1 to 100	10
7	SNR	5 to 500	50

5

SECTION

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 problem for the following objectives:

1. To find the connectivity of unknown links in a conserved network by feeding in flow data and connectivity of known links
2. To generate a the graph of a network based on a particular network model and pertaining flow data, and then let the code find connectivity of unknown links through the incidence matrix from that data.

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

5.1 Feed Data → Find Topology

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

1. Data Matrix in variable named 'X' with exactly e rows and any number of columns
2. Incidence information of known links in variable named 'Ab' and corresponding edges as a row vector in variable named 'Ab_edges'. Note that 'Ab' and 'Ab_edges' should have the same number of columns.
3. In case of noisy data, if it is homoscedastic or heteroscedastic

4. 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', 'Ab', 'Ab_edges' 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. Program checks for variable 'Ab' in the workspace and terminates the program if not available
6. Program checks for variable 'Ab_edges' in the workspace and terminates the program if not available
7. Program checks if the columns of 'Ab' and 'Ab_edges' are equal and terminate the program if not
8. Choose whether the data is 'noisy' or 'noisefree', when asked
9. In noisy case, choose whether the noise is 'homoscedastic' or 'heteroscedastic', when asked
10. In heteroscedastic case, program checks for variable 'SNR' in the workspace and terminates the program if not available
11. Finally, the code runs and gives an output whether some or all of the unknown links are possibly identified. The time taken is also mentioned.

5.2 Randomly Generate Network & Data → Find 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 nodes 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

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

6. SNR values 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 - Erdos-Renyi, 2 - Watts–Strogatz or 3 - Barabasi-Albert, when asked or Erdos-Renyi network is taken by default

4. Choose the number of nodes in the network in the given range or default value of 10 is taken

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 a value for noise variance to be added or a default value of 10 is taken

9. In heteroscedastic case, choose an SNR value for noise to be added

10. In noisy case, choose the number of datasets to be generated for a network or a default value of 10 is taken

11. Finally, the code runs and gives an output whether the topology could be identified or not including the average time taken

SECTION 5

References

[1] Satya Jayadev, P., Narasimhan, S., and Bhatt, N. (2019). Learning Conserved Networks from Flows. arXiv preprint 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.

[10] Erdös, P and Rényi, A (1959). On random graphs i. Publ. Math. Debrecen, 6(290-297), 18.

[11] Watts, D. J., and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. nature, 393(6684), 440.

[12] Albert, R., and Barabási, A. L. (2002). Statistical mechanics of complex networks. Reviews of modern physics, 74(1), 47.

[13] Pablo Blinder (2020). Erdos-Renyi Random Graph (https://www.mathworks.com/matlabcentral/fileexchange/4206-erdos-renyi-random-graph), MATLAB Central File Exchange. Retrieved March 3, 2020.

[14] Tapan (2020). Scale free network using B-A algorithm (https://www.mathworks.com/matlabcentral/fileexchange/49356-scale-free-network-using-b-a-algorithm), MATLAB Central File Exchange. Retrieved March 3, 2020.