Order Reduction of Large Scale Circuit Graphs

Diploma Thesis

By

Skouroliakou Vasiliki

Department of Electrical and Computer Engineering
University of Thessaly

Supervisors
Georgios Stamoulis, Professor
Tsompanopoulou Panagiota, Assistant Professor
Potamianos Gerasimos, Assistant Professor

September 2019

Περίληψη

Τις τελευταίες δεκαετίες υπάρχει ένα αυξανόμενο ενδιαφέρον για την κατασκευή μαθηματικών μοντέλων τα οποία περιγράφουν σύνθετα συστήματα. Συγκεκριμένα, όσον αφορά τον τομέα του σχεδιασμού κυκλωμάτων τα μαθηματικά μοντέλα δημιουργούνται με σκοπό να χρησιμοποιηθούν σε αριθμητικές προσομοιώσεις, οι οποίες μας βοηθούν να αναπαράξουμε τη συμπεριφορά του κυκλώματος. Ωστόσω, τα μαθηματικά μοντέλα, έχοντας προκύψει από την ανάλυση ρεαλιστικών κυκλωμάτων, γίνονται ολοένα και πιο πολύπλοκα οδηγώντας σε μοντέλα μεγάλων διαστάσεων, τα οποία εντέλει αδυνατούμε να επεξεργαστούμε αριθμητικά.

Επομένως, ένα μεγάλο κομμάτι έρευνας έχει αφιερωθεί στην ανάπτυξη τεχνικών που έχουν ως στόχο τη μείωση της διάστασης των μοντέλων σε μια μικρότερη, ώστε να είναι πλέον δυνατή η χρήση τους σε αριθμητικές προσομοιώσεις. Αυτη η μέθοδος προσέγγισης ονομάζεται ”Μείωση Τάξης Μοντέλου”. Μια μέθοδος για να καταλήξουμε στο μειωμένης τάξης μοντέλο από ένα αρχικό μεγάλης τάξης, προτείνεται από τους Monshizadeh, Harry L. Trentelman και M. Kanat Camlibel σε σχετικό άρθρο τους το οποίο δημοσιεύτηκε το 2014 από τον οργανισμό IEEE. Η μέθοδος βασίζεται στην κατάτμηση των κορυφών ενός γράφου σε μικρότερα τμήματα, κάθε ένα από τα οποία θεωρείται ένας καινούριος κόμβος στο μειωμένο μοντέλο.

Στόχος της παρούσας διπλωματικής είναι -έχοντας ένα αρχικό μοντέλο που προήλθε από ένα RC δίκτυο- να παράξουμε ένα μειωμένης τάξης μοντέλο βασιζόμενοι στην παραπάνω μέθοδο. Θα περιγράψουμε αναλυτικά τη διαδικασία που ακολουθείται και κλείνοντας θα παρουσιάσουμε τη σύγκριση μεταξύ αρχικού και μειωμένου μοντέλου.

Abstract

In the last few decades there is an increasing interest in creating mathematical models to describe complex physical systems. Especially, concerning circuit design area mathematical models are mostly generated to be used in simulations that help us replicate the behavior of a circuit. However, circuit designing process becomes more and more complicated nowadays forcing mathematical models to adopt that complexity. As a result, it is too hard to use them in simulations.

Therefore, a lot of research has been devoted to developing techniques in order to reduce the high order of a model to a lower one, such that it is feasible to deal with it numerically. These approaching techniques are generally called ”Model Order Reduction”. One technique that aims to obtain a reduced order model is proposed by N.Monshizadeh, Harry L. Trentelman and M. Kanat Camlibel in their work published in 2014 by IEEE. The method is based on partitioning a set of nodes in a graph into cells, called clusters, and then regard each cluster as a single node in reduced model.

The objective of this thesis is to obtain a reduced order model for a given original one, based on the above method. The original model has derived from a RC network. We will describe the procedure analytically and in closing we will present the comparison between the original and reduced model.

Dedication and Acknowledgements

Firstly, i would like to thank my lead supervisor, professor Nestor Eumorfopoulos, for all his help and guidance to complete this thesis. I would also like to thank my family and friends for their support during all this years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modeling large scale, complex dynamical systems is one of grate importance problems in engineering. However, most of the times the models extracted directly from physical systems analysis are too complicated to be used in numerical simulations. This could be either due to limitations in computational resources or because the simulation requires a significant amount of time or storage capatiance. Consider the electronic industry and specifically the design of computer chips [6]. Their development is highly associated with Moore's law, which generally states that every 18 months a new generation of chips is created. Every new generation is twice as fast as the previous, operating at higher frequencies while, at the same time the dimension of devices are dramatically reduced. That means that chip design becomes more and more complex. Simulating the behavior of such a circuit along with interconnection structure consists a huge mathematical problem with high complexity. Indicatively, the scale of the model could be 10^{11}.

So, the question is whether a simpler model of the system can be obtained. Model order reduction techniques aim to answer this question.[8]. The general idea is to replace the large scale model of the system with one of lower dimension. The reduced model should behave similarly to the original one concerning its frequency or time response characteristics. This thesis focuses on linear time invariant systems modeling, meaning that the original system is described by a set of linear differential equations with constant coefficients. There are two types of linear order reduction techniques: those based on projection and those based on truncation techniques.

In this thesis a projection method based on graph partitioning is described. We consider the Laplacian matrix of the system to be a graph and we aim to obtain a partition of it. The projection is formed in terms of the characteristic matrix of the partition. The advantage of this procedure is that the spatial structure of the system is preserved, meaning that the reduced model is still an input-output system. The objective is to compare the behavior of the reduced and original model. This thesis is organized in chapters as seen in the following Table 1.1:

Chapter 1	Introduction
Chapter 2	Linear Time Invariant Systems and Circuit Modeling
Chapter 3	Graph Theory and Graph Partitioning
Chapter 4	Model Order Reduction
Chapter 5	Model Reduction of RC Network
Chapter 6	Numerical Results
Chapter 7	Conclusions

Table 1.1: Chapter overview

Chapter one is an introductory chapter. Chapter two introduces linear time invariant systems and also describes the procedure to extract a linear time invariant model from a simple circuit using modified nodal analysis method. Chapter three is an introduction to graph theory and exhibits the concept of graph partitioning. Chapter four presents a projection-based model order reduction technique that uses the partition of the graph to form projection matrices. Chapter five applies the previous technique to a certain model originated from a RC network. We use two alternative methods to obtain a partition of the graph. Chapter six includes numerical results concerning the comparison between the reduced and original model in frequency and time domain. Finally, chapter seven concludes this thesis.

Chapter 2

Linear Time Invariant Systems (LTI) and Circuit Modeling

Mathematical models, used to describe physical systems, are many times in the form of linear time invariant systems. They consist of a set of first order differential equations in terms of a sate vector x(t). A classical paradigm arises from RLC circuit area. In this chapter, after introducing linear time invariant systems, we will use modified node analysis method to extract the mathematical model for a simple RC circuit.

2.1 Definition

Linear Time Invariant Systems are a special category of systems which exhibit two basic properties[9],[19]:

- Linearity: The main aspect based on which a system is considered to be linear or not is superposition. Superposition principal states that if the input applied to a system is the sum of two or more individual signals, then the overall response of the system is the sum of the responses that each individual signal would have caused. Specifically, if a system is described by f(x), and $y_1 = f(x1)$, $y_2 = f(x2)$ are the responses corresponding to input signals x_1 and x_2 respectively, then the system is linear if the following conditions are satisfied:

  1. $f(x1+x2) = f(x1) + f(x2) = y1 + y2$
  2. $f(ax) = af(x) = ay$,for any scalar factor a.

- Time Invariance: A system is considered to be time invariant if a shift in time on the input causes the same shift in time on the output. Specifically, if $y(t) = x(t)$, then $y(t-\delta) = x(t-\delta)$.

2.2 Differential Equations and State Variables

2.2.1 State Variables

State variables refer to a minimum set of system variables enough to fully describe the state of the system mathematically and also determine the response of the system to any given input.[15]. Knowing these variables $\{x_1, x_2, ..., x_n\}$ and their initial value at a time t_0, together with the system inputs for time $t \geqslant t_0$, suffices to predict any future state of the system and, in addition, the output vector for time $t > t_0$.

For example, the system in Figure 2.1 has an input vector of size m, $\{u_1, u_2, ...u_m\}$ and an output vector of size q, $\{y_1, y_2, ..., y_q\}$. The knowledge of state variables $\{x_1, x_2, ..., x_n\}$ and their value at time t_0, as well as the knowledge of the input vector $\{u_1, u_2, ..., u_m\}$ for time $t \geqslant t_0$ are all we need to determine any future behavior of the system and compute the output vector $\{y_1, y_2, ..., y_q\}$.

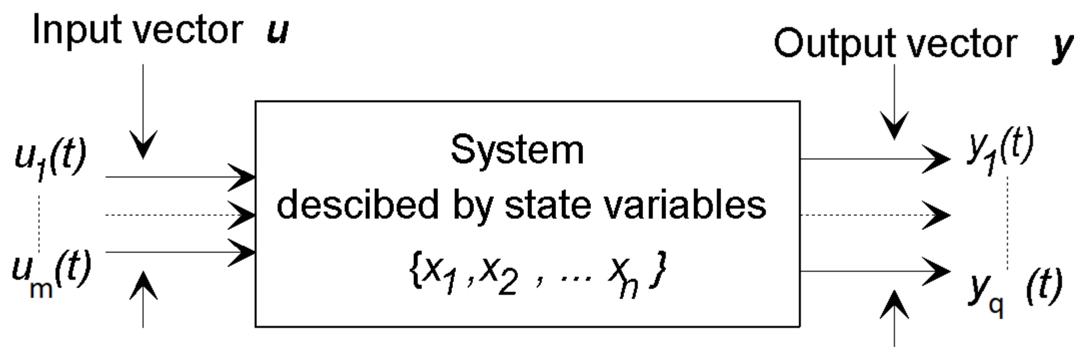


Figure 2.1: State Variables from [15]

No set of state variables, used to describe a system is unique. In fact, any set of system variables, able to describe the system effectively can be considered to be state variables. On the other hand, the order of a given system is unique, independent of the chosen set of state variables and equal to n.

2.2.2 Differential Equations

Mathematically, a system is described by a set of n coupled first order differential equations. Each one of them expresses the time derivative of states variables in terms of state variables $\{x_1, x_2, ...x_n\}$ and system inputs $\{u_1, u_2, ..., u_m\}$. Generally a state equations is in the form:

$$x_i' = f_i(x, u, t) \tag{2.1}$$

$i = \{1, 2, ..., n\}$,where $x' = \frac{dx}{dt}$ and f could be a nonlinear time varying function in terms of state variables, inputs and time.

Usually, the sets of n state variables and m inputs are expressed in vector form, $x(t) = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}^T$ and $u(t) = \begin{bmatrix} u_1 & u_2 & ... & u_m \end{bmatrix}^T$, so the set of state equations is written:

$$x' = f(x, u, t) \tag{2.2}$$

referring to x' and f as vectors with n components.

In case of linear time invariant systems the differential equations are linear with constant coefficients:

$$x_1(t)' = a_{1_1}x_1(t) + a_{1_2}x_2(t) + ...a_{1_n}x_n(t) + b_{1_1}u_1 + ... + b_{1_m}u_m \tag{2.3a}$$
$$x_2(t)' = a_{2_1}x_1(t) + a_{2_2}x_2(t) + ...a_{2_n}x_n(t) + b_{2_1}u_1 + ... + b_{2_m}u_m \tag{2.3b}$$
$$... \tag{2.3c}$$
$$... \tag{2.3d}$$
$$... \tag{2.3e}$$
$$x_n(t)' = a_{n_1}x_1(t) + a_{n_2}x_2(t) + ...a_{n_n}x_n(t) + b_{n_1}u_1 + ... + b_{n_m}u_m \tag{2.3f}$$

Equations (2.3) can be written in matrix form aw well:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & . & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & . & a_{2n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{n1} & a_{n2} & . & . & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & . & . & . & b_{1m} \\ b_{21} & b_{22} & . & . & . & b_{2m} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ b_{n1} & b_{n2} & . & . & . & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_m \end{bmatrix} \tag{2.4}$$

which may be summarized in equation:

$$x'(t) = Ax(t) + Bu(t) \tag{2.5}$$

where x is a column vector of size n containing the state variables, u is a column vector of size m containing the system inputs, A is a square $n \times n$ matrix of constant coefficients and B is an $n \times m$ matrix consisting of the coefficients that weight the inputs.

The output of the system consists of any variables of interest. One output variable y_i is expressed as a linear combination of state variable x_i and system input u_i. One arbitrary output variable could be:

$$y_i(t) = c_1x_1(t) + c_2x_2(t) + ... + c_nx_n(t) + d_1u_1 + ... + d_mu_m \tag{2.6}$$

where c_i, d_i are constants. If the output vector of a given system is of size q, then q equations are constructed in the form of (2.6):

$$\begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_q \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} & . & . & . & c_{1n} \\ c_{21} & c_{22} & . & . & . & c_{2n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ c_{q1} & c_{q2} & . & . & . & c_{qn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & d_{21} & . & . & . & d_{1m} \\ d_{21} & d_{22} & . & . & . & d_{2m} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ d_{q1} & d_{q2} & . & . & . & d_{qm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_m \end{bmatrix} \tag{2.7}$$

which could be summarized in equation:

$$y = Cx(t) + Du(t) \tag{2.8}$$

where y is a column vector of size q containing the output variables, C is an $q \times n$ matrix of constant coefficients and D is a $q \times m$ matrix of constant coefficients that weight the system inputs. For many systems the matrix D is a null matrix, so the output variables are written as a combination of state variables only.

$$y = Cx(t) \tag{2.9}$$

So, the complete model used to describe a linear time invariant system is summarized in equations (2.5) and (2.8). Matrices A and B result from the structure of the system, while C and D result from the particular choice of output variables.

2.3 Transfer Function of LTI Systems

In previous lines we described how LTI systems are represented in time domain using state equations. Alternatively, any LTI system can be represented in frequency domain by computing transfer function H(s) using Laplace Transform.

To begin with, we quote two basic properties of Laplace Transform, concerning the transfer of a function f(t) and the first derivative $f'(t)$ to s-domain:

$$\mathscr{L}\{f(t)\} = F(s) \tag{2.10}$$

$$\mathscr{L}\{f'(t)\} = sF(s) - f(0) \tag{2.11}$$

So using the Laplace Transform and the initial condition $x(0) = 0$, equations (2.5) and (2.8) are written in s-domain:

$$\mathscr{L}\{x' = Ax + Bu\} \Longrightarrow sX(s) = AX(s) + BU(s) \tag{2.12}$$

$$\mathscr{L}\{y = Cx + Du\} \Longrightarrow Y(s) = CX(s) + DU(s) \tag{2.13}$$

In the case that D is a null matrix the second one is written:

$$\mathscr{L}\{y = Cx\} \Longrightarrow Y(s) = CX(s) \tag{2.14}$$

Transfer function H(s) is defined as a fraction with numerator equal to the output of the system in s-domain and denominator equal to the input applied to it in s-domain too:

$$H(s) = Y(s)/U(s) \tag{2.15}$$

So we need to rewrite (2.12) as an expression of X(s) and combine it with (2.14) to compute Y(S) in terms of U(s):

$$sX(s) = AX(s) + BU(s) \tag{2.16a}$$

$$sX(s) - AX(s) = BU(s) \tag{2.16b}$$

$$(sI - A)X(s) = BU(s) \tag{2.16c}$$

$$X(s) = (sI - A)^{-1}BU(s) \tag{2.16d}$$

and (2.14) is now written:

$$Y(s) = C(sI - A)^{-1}BU(S) \tag{2.17}$$

According to (2.15) transfer function is computed:

$$H(s) = C(sI - A)^{-1}B \tag{2.18}$$

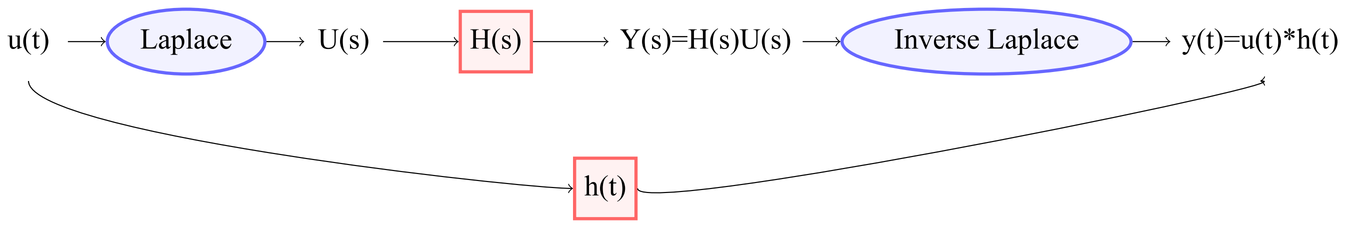


Figure 2.2: LTI Systems in Time and Frequency Domain

2.4 Circuit Modeling

2.4.1 Circuit Structure

Every electronic circuit consists of basic elements which are: ohmic resistor, capacitor, inductor, voltage and current sources [4],[16]. In fact, real physical circuits contain semiconductor devices too, which for modeling purposes are replaced by idealized elements. Each basic element of the circuit corresponds to a characteristic equation:

- Resistor: $I = \frac{1}{R}V$
- Capacitor: $I = CV'$
- Inductor: $V = LI'$

- Current Source: $I = i(t)$
- Voltage Source: $V = v(t)$

In addition, again for modeling purposes, all wires between circuit elements are considered to be ideal, meaning they have no extra resistance, capatiance or inductance. Hence, the circuit is reduced to a conceptual structure that contains only the basic elements connected by ideal wires. This conceptual structure is usually called network topology.

In conceptual form, a circuit is an assemblage of branches and nodes. A branch originates from a circuit element while the terminals of the element are the nodes, which allow the interconnection with other elements. We assign a direction to each branch according to the direction of current flow through it and a serial number to each node. Usually one node of the circuit is regarded as the ground node, meaning that has constantly zero voltage and as a consequence all node voltages are expressed as voltage differences between each node and the ground node.

The electrical circuit shown in Figure 2.3 consists of five branches- two resistors, two capacitors and a voltage source- and four nodes including the ground node. In this example ground node is the node denoted by number four. The circuit is a second order RC filter.

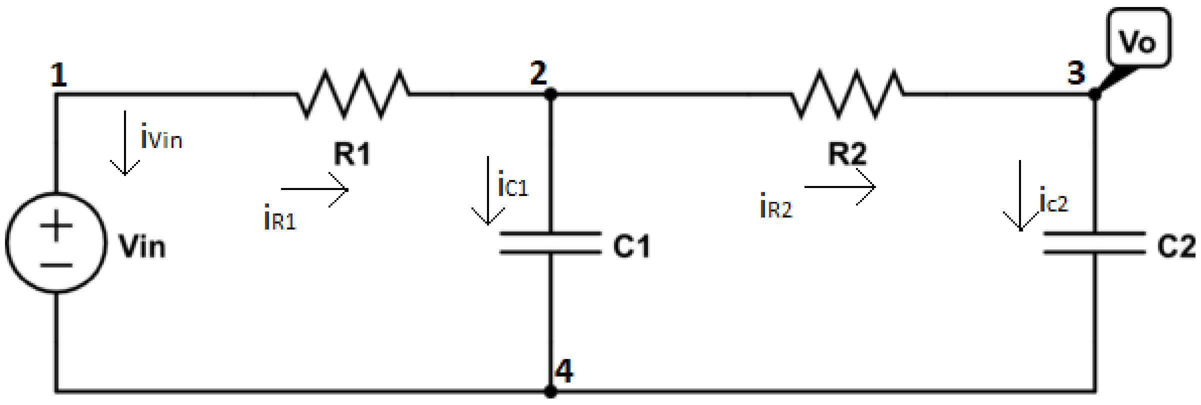


Figure 2.3: Second Order RC Filter

2.4.2 Incidence Matrix

For any circuit in the conceptual form we can define the incidence matrix associated with it. This matrix is $p \times b$, where p is the number of nodes, including the ground node, and b is the number of branches. The elements of the matrix are defined as follows taking into account the current flow through each branch:

$$r_{ij} = \begin{cases} -1 : \textbf{if node i is the tail of branch j} \\ 1 : \textbf{if node i is the head of branch j} \\ 0 : \textbf{otherwise} \end{cases} \tag{2.19}$$

For the circuit shown in Figure 2.3 the incidence matrix will be 4×5, cause it consists of five nodes and four branches:

$$R = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ -1 & 0 & -1 & 0 & -1 \end{bmatrix} \tag{2.20}$$

Since the voltage of the ground node is constantly zero the information concerning that node is redundant, so the incidence matrix can be reduced to $(p-1) \times b$. In our example the incidence matrix is the same with the one in (2.20) removing the last row, hence it is 3×5.

We could also define individual incidence matrices for every basic element of the circuit, concerning only the branches associated with this particular element. The basic elements of a circuit are: resistor or conductance(R or G),capacitor(C), inductor(L), voltage source(V) and current source (I), so the available matrices to be defined are: $R_{R,G}$, R_C, R_L, R_V, R_I. For the circuit in Figure 2.3 we define three extra matrices: one for capacitors, one for resistors or conductances and one for the voltage source. These matrices are $(p-1) \times b$, where b now indicates the number of branches including one certain element and b is equal to the number of nodes excepting the ground one. The capacitors matrix is 3×2, as long as the circuit consists of three nodes- excepting the ground node- and two capacitor branches. The other two matrices are formed correspondingly.

$$R_C = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, R_G = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, R_V = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.21}$$

2.4.3 Modified Nodal Analysis Method

We are going to use Modified Nodal Analysis (MNA) method in order to obtain a mathematical model for the circuit shown in Figure 2.3. The method aims to determine the node voltages of a circuit applying the Kirchhoff's current law at every node. The current law states that the total current entering a node is exactly equal to the total current leaving the same node or in other words the sum of currents meeting at a certain node is equal to zero. Conventionally, the current leaving one node is regarded as "positive". The example circuit has three nodes to apply Kirchhoff's current law as the ground node is excepted. In addition, we are going to use the characteristic equations of the elements found in the circuit ,as well as the fact that ohmic resistor is commonly expressed in terms of conductance as $G = \frac{1}{R}$.

The currents of the branches are i_{R1}, i_{R2}, i_{C1}, i_{C2}, i_{Vin} and the current flow direction through them is shown in the picture 2.3. Kirchhoff's current law, applied at nodes e_1,e_2, e_3, yields:

$$i_{V_{in}} + i_{G_1} = 0 \tag{2.22a}$$

$$-i_{G1} + i_{C1} + i_{G2} = 0 \tag{2.22b}$$

$$-i_{G2} + i_{C2} = 0 \tag{2.22c}$$

Looking at the circuit, we easily observe that the node voltages e_1,e_2, e_3 are associated with element voltages as follows:

$$V_{in} = e_1 \tag{2.23a}$$

$$V_{R1} = e_1 - e_2 \tag{2.23b}$$

$$V_{C1} = e_2 \tag{2.23c}$$

$$V_{R2} = e_2 - e_3 \tag{2.23d}$$

$$V_{C2} = e_3 \tag{2.23e}$$

Using the characteristic equations $I = GV$ and $i_C = C\frac{d_V}{dt}$ the above equations are written:

$$i_{V_{in}} + (e_1 - e_2)G1 = 0 \tag{2.24a}$$

$$-(e_1 - e_2)G1 + C_1\frac{de_2}{dt} + (e_2 - e_3)G2 = 0 \tag{2.24b}$$

$$-(e_2 - e_3)G2 + C_2\frac{de_3}{dt} = 0 \tag{2.24c}$$

However, note that there are three equations for four unknowns: e_1,e_2, e_3, $i_{V_{in}}$. The fourth equation needed is the fact that:

$$V_{in} = e_1 \tag{2.25}$$

As long as all the elements in the circuit obey to linear characteristic equations we can write equations (2.24), (2.25) in the form:

$$Ex'(t) = -Ax(t) + Bu(t) \tag{2.26}$$

The four unknown variables comprise the column vector $\in R^4$:

$$x(t) = [e_1 \;\; e_2 \;\; e_3 \;\; i_{Vin}]^T \tag{2.27}$$

The input vector u consists of the excitation, meaning the voltage source V_{in}, applied to node e_1, so $u \in R$:

$$u(t) = [V_{in}] \tag{2.28}$$

Matrices E, A and B are directly extracted from equations (2.24), (2.25). Matrix E concerns the time derivative $x'(t)$, while matrix A contains the coefficients of x(t) and matrix B associates the excitation V_in with equation (2.25).

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & 0 & 0 \\ 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, A = \begin{bmatrix} G1 & -G1 & 0 & 1 \\ -G1 & G1+G2 & -G2 & 0 \\ 0 & -G2 & G2 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.29}$$

The output is usually a linear relationship of state vector and the input, generally of the form:

$$y(t) = Cx(t) + Du(t) \tag{2.30}$$

However, in circuit design we rarely observe feedthrough of the input to the output, hence matrix D is equal to zero. Moreover, the matrix C is often the transpose of the input matrix B, B^T. Therefore, the output vector y is written:

$$y(t) = B^T x(t) \tag{2.31}$$

where in our example $B^T = [0\ \ 0\ \ 0\ \ 1]$.

Generally, Modified Nodal Analysis method can be applied to any circuit, in order to write it in the form of (2.26) and (2.31). Consider a circuit having p nodes- including the ground node- and b branches. The number of unknown variables and therefore the size of the state vector x(t) is equal to $n = (p-1)+m$, where m is the sum of sources, voltage or current. We apply Kirchhoff's current law to nodes $\{e_1, e_2, ..., e_{p-1}\}$- ground node is excepted. That results to $p-1$ equations, containing the unknown variables. We find the extra m equations needed using the fact that every source is assigned to a node e_i. We should take into account that the circuit may contain inductors too. This makes state vector x(t) to grow, since the current i_L flowing through each inductor branch is an unknown variable too. Moreover, matrix E should contain this extra information about inductors.

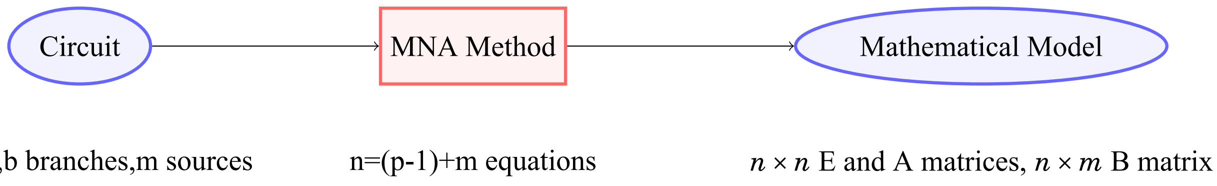

Circuit
MNA Method
Mathematical Model
p nodes,b branches,m sources
n=(p-1)+m equations
$n \times n$ E and A matrices, $n \times m$ B matrix

Figure 2.4: Modified Nodal Analysis Method

The matrices E, C and B of can be extracted directly from individual incidence matrices as follows:

$$E = \begin{bmatrix} R_C C R_C^T & & 0 \\ 0 & L & 0 \\ 0 & 0 & 0 \end{bmatrix}, A = \begin{bmatrix} R_G G R_G^T & R_L & R_V \\ -R_L^T & 0 & 0 \\ -R_V^T & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} R_I & 0 \\ 0 & 0 \\ 0 & I_{n_v} \end{bmatrix} \tag{2.32}$$

The C, G and L are diagonal matrices containing the values of capacitors, conductances and inductors of the circuit respectively. The I_{n_v} is the identity matrix in $R^{n_v \times n_v}$, where n_v is the number of voltage sources in the circuit.

Here, E and A $R^{n \times n}$ and B $R^{n \times m}$.

Chapter 3

Graph Theory and Graph Partitioning

Generally, a graph is a structure consisting of a set of objects that are somehow related to each other.In this chapter we will introduce graph theory and define the basic matrices associated with graphs. In addition, we will describe the concept of graph partitioning.

3.1 Graph Definition

A simple graph consists of a finite set of elements, which is called the vertex set and it is denoted by V [12]. Each element in this set $V = \{v_1, v_2, ...v_n\}$ is a vertex in the graph. The graph is formally defined by the pair $G = (V, E)$, where E comprises the edge set of the graph. Usually one edge is denoted by the letter e and two pointers $\{ij\}$ indicating the connection between vertices $\{v_i, v_j\}$. The graphical representation of a graph consists of ”dots” indicating vertices and ”lines” indicating the connection between two vertices $\{v_i, v_j\}$ when $\{i, j\} \in E$.

A graph is called undirected, when every $\{v_i, v_j\} \in E$ simply connects vertices $\{v_i, v_j\}$, without indicating direction from one vertex to the other. Alternatively, it is called directed, when each edge $\{eij\} \in E$ is actually an arc starting from the head vertex v_i and ending at the tail one v_j. Furthermore, a graph is considered to be weighted, if a non-negative number -called weight- is assigned to each edge, otherwise it is considered to be unweighted.

In the following Figure 3.1 a graph of five vertices $V = \{u_1, u_2, u_3, u_4, u_5\}$ and six edges $\{e_{12}\}, \{e_{23}\}$, $\{e_{34}\}, \{e_{35}\}, \{e_{25}\}, \{e_{45}\}$ is shown:

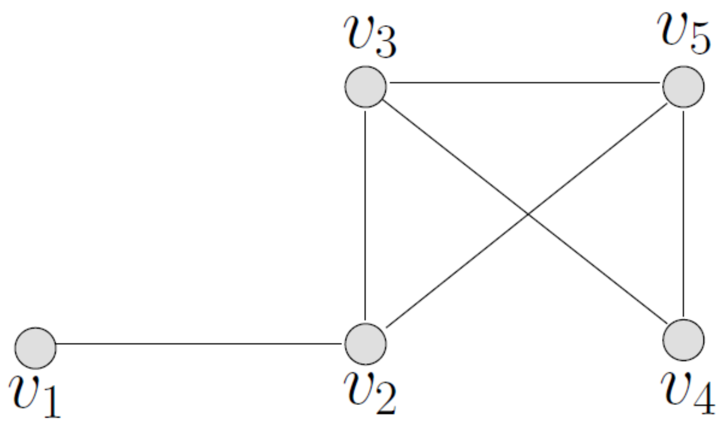


Figure 3.1: An undirected and unweighted graph from [12]

3.2 Graphs and Matrices

3.2.1 Degree Matrix

For an undirected, unweighted graph the Degree matrix is a diagonal matrix of size $n \times n$, where n is equal to number of the vertices in the graph. For each vertex, the corresponding diagonal element $d(v_i)$ is equal to the number of vertices adjacent to vertex v_i. In case of weighted graphs the diagonal elements $d(v_i)$ of Degree matrix are equal to the sum of weights of edges adjacent to each vertex v_i.

The Degree matrix of the above graph 3.1 is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} \tag{3.1}$$

3.2.2 Adjacency Matrix

The Adjacency matrix A is a symmetric $n \times n$ matrix which represents the connections between the vertices of the graph. The elements of Adjacency matrix are defined:

$$|a_{ij}| = \begin{cases} 1: \textbf{if vertices } v_i, v_j \textbf{ are connected} \\ 0: \textbf{otherwise} \end{cases} \tag{3.2}$$

In case of weighted graphs:

$$|a_{ij}| = \begin{cases} w_{ij}: \textbf{if vertices } v_i, v_j \textbf{ are connected} \\ 0: \textbf{otherwise} \end{cases} \tag{3.3}$$

For the graph in 3.1 the Adjacency matrix is:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.4}$$

3.2.3 Incidence Matrix

We referred to Incidence matrix R in Chapter two too. It is an $n \times m$ matrix, where n is the number of vertices in the graph and m the number of edges. The incidence matrix encodes not only the adjacency of the vertices in the graph but also the orientation of edges, thus it concerns only directed graphs. The elements of incidence are defined:

$$r_{ij} = \begin{cases} -1 : \textbf{if vertex } v_i \textbf{ is the tail of edge } e_{ij} \\ 1 : \textbf{if vertex } v_i \textbf{ is the head of edge } e_{ij} \\ 0 : \textbf{otherwise} \end{cases} \tag{3.5}$$

3.2.4 Laplacian Matrix

An extra representation of a graph is Laplacian matrix which can be extracted directly from Degree and Adjacency matrices as follows:

$$L = D - A \tag{3.6}$$

For example, the Laplacian matrix corresponding to the graph shown in 3.1 is:

$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix} \tag{3.7}$$

Alternatively, given an arbitrary orientation to the edge set, Laplacian matrix can be defined using Incidence matrix as:

$$L = RR^T \tag{3.8}$$

If the graph is weighted then the Incidence matrix is defined:

$$L = RWR^T \tag{3.9}$$

where W is an $n \times n$ diagonal matrix containing the weights $w(e_{ij})$ on the diagonal.

3.3 From Circuit to its Graph

It is common to represent circuits using their equivalent graph. To define a graph $G = (V,E)$ we need to identify the set of vertices $V = \{v_1, v_2, ..., v_n\}$ and the set of edges $E = \{e_{ij}\}$, which are mapped to the nodes and branches of the circuit respectively. As long as, we have assigned a direction to the current flowing through each branch the graph will be directed. For the RC filter shown in Figure 2.3, the corresponding graph is:

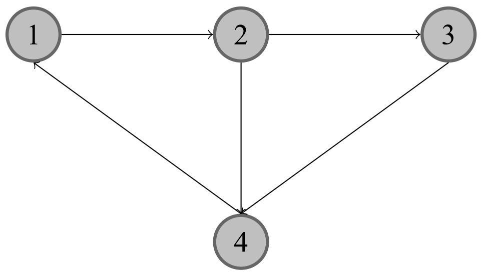


Figure 3.2: Graph corresponding to circuit in Figure 2.3

3.4 Graph Partitioning

Computer scientists frequently use graphs when modeling real life problems. Cutting a graph into smaller pieces is often part of the solution, making graph partitioning a major algorithmic problem with a lot of applications. When speaking of graph partitioning we refer to dividing a large graph into two or more individual clusters- also called cells- based on a certain condition.

3.4.1 Characteristic Matrix of Partition

Let $G = (V,E)$ be a graph and $V = \{v_1, v_2 ...., v_n\}$ the set of vertices. Any non-empty subset of V, comprises a cell of V. A collection of cells, $\pi = \{C_1, C_2, ...C_r\}$ is called a partition of V if $\cup i C_i = V$ and $C_i \cap C_j = \emptyset$ whenever $i \neq j$ [13]. We consider π to be a partition of G as well. In addition, we define a column vector, indicating which vertices belong to each particular cell $C \subseteq V$:

$$P(C_r) = \begin{cases} 1 : \textbf{if vertex i belongs to } C_r \\ 0 : \textbf{otherwise} \end{cases} \tag{3.10}$$

For a certain partition $\pi = \{C_1, C_2, ..., C_r\}$ of V the individual column vectors are combined to form the characteristic matrix $P(\pi)$ or simply P. Matrix P is $n \times r$, where n is the original number of vertices in graph and r is the number of cells.

The objective here is to create a graph having fewer vertices than the original one. If the original graph G has n vertices, the reduced one $\tilde{G}$ should have r«n vertices, where r is equal to the number of cells in the partition. Each cell of the partition π in G is mapped to a new vertex in $\tilde{G}$. Moreover, there is an arc from vertex p to vertex q in $\tilde{G}$ if and only if there exists $i \in C_p$ and $j \in C_q$ with $p \neq q$ such that $e_{ij} \in E$.

Let's take the graph shown in the following Figure 3.3 as an example. The graph consists of ten vertices and we aim to obtain a partition π such that the reduced graph $\tilde{G}$ consists of five vertices.

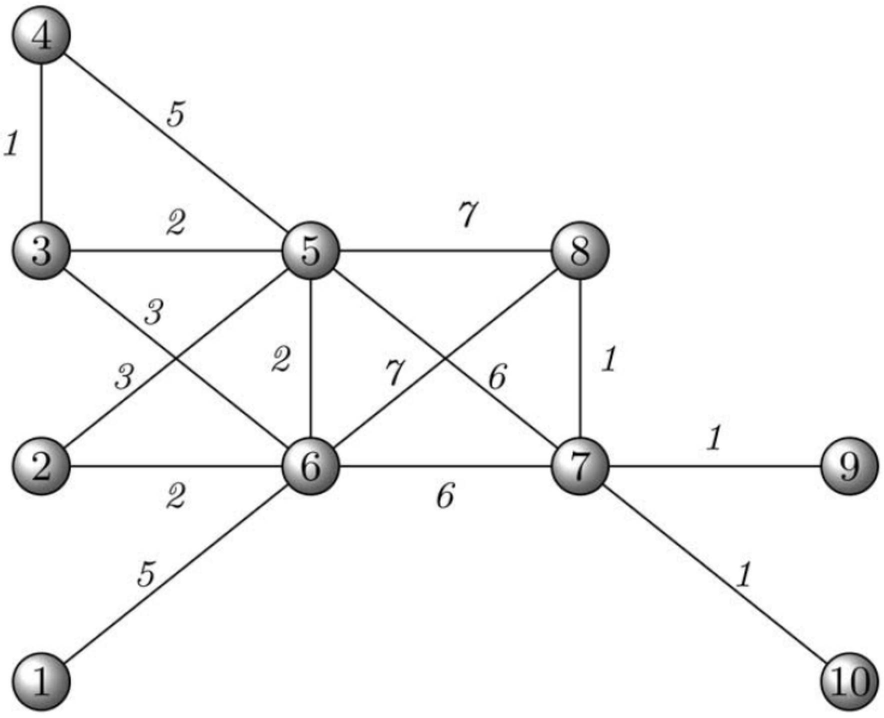


Figure 3.3: G=(V,E) from [13]

The partition chosen is:

$$\pi = \{\{1,2,3,4\},\{5,6\},\{7\},\{8\},\{9,10\}\} \tag{3.11}$$

Hence, the characteristic matrix is:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

3.4.2 Reduced Graph

The reduced graph $\tilde{G}$ will be:

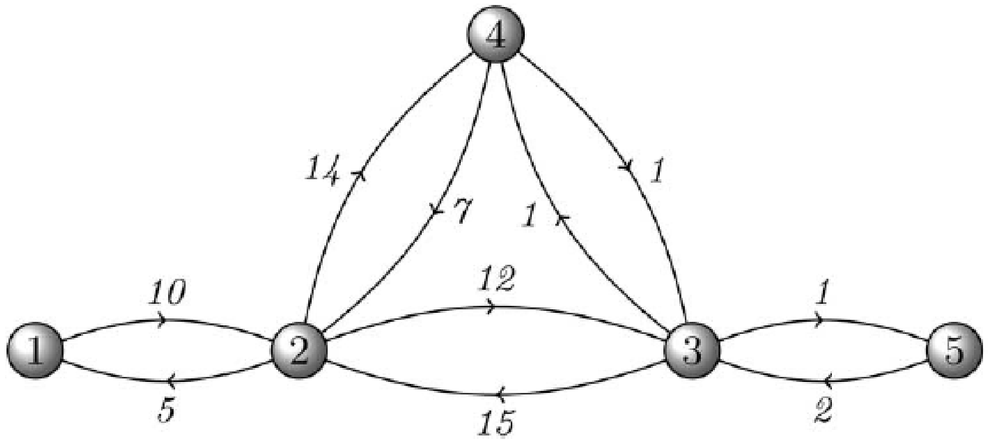


Figure 3.4: $\tilde{G}$=($\tilde{V}$,$\tilde{E}$) from [13]

Graph $\tilde{G}$ consists of five vertices, one for each cell in partition π. For instance, vertex one in $\tilde{G}$ corresponds to the cell $C_1 = \{1,2,3,4\}$ and similarly vertex two correspond to cell $C_2\{5,6\}$. Concerning edges $\{e_{ij}\} \in \tilde{E}$ the weights associated with them are equal to the average of the weights of the edges $\{e_{ij}\} \in E$ with $i \in C_p$ and $j \in C_q$. They are given by the relationship:

$$w_{pq} = \frac{1}{C_q} \sum_{i \in C_p, j \in C_q} w_{ij} \tag{3.13}$$

where p,q denote two cells in G and therefore two vertices in $\tilde{G}$.

3.5 Graph Partitioning Techniques

This section goes through some of the most popular techniques trying to solve graph partitioning problem [7],[14].

- Spectral Partitioning: Spectral Partitioning is one of the first methods used to partition a graph. Originally the method aimed to split a graph into two clusters (spectral bisection) by computing the eigenvector associated with the second smallest eigenvalue of the Laplacian matrix of the graph. Then the median value of this eigenvector is determined and based on that the two clusters are created. The first one consists of all nodes with an entry smaller or equal with median value and the second one consists of the rest of the nodes. Later, the method was expanded to produce

$k > 2$ clusters by using multiple eigenvectors. The method is more useful when a small number of clusters is required, namely four or eight.

- Breadth First Search: This is a very simple method that works as follows: Starting from a random vertex u, the method traverses the graph using Breadth First Search algorithm, assigning the visited vertices to cluster V_1. When half of the original vertices are assigned to V_1 the method stops. Then the remaining vertices are assigned to cluster V_2.
- Kernighan-Lin (KL) and Fiduccia-Mattheyes Algorithms: Kernighan-Lin is one of the oldest graph partitioning algorithms. It is a local search algorithm, which given a graph $G = (V,E)$ and a two-way or k-way partition of it, tries to produce a new optimum partition, improving the cut size. Kernighan-Lin is also a balanced partitioning algorithm, meaning that it aims to produce equal size clusters of vertices. It iteratively exchange pair of nodes between clusters of the partition, until it reaches an optimum one. Fiduccia-Mattheyes algorithms are an improvement of Kernighan-Lin algorithm. The difference is that Fiduccia-Mattheyes method selects a single node to exchange between clusters in each iteration.
- Multilevel Graph Partitioning: This method is comprised of three phases [18]:
  1. Coarsening Phase: At this phase a sequence of $G_1, G_2, ..., G_m$ smaller graphs are extracted from original graph G. Each vertex in one of smaller graphs represent a set of connected vertices in original graph G. At each stage of this phase a set of interconnected vertices are combined to comprise a vertex in the next stage. The coarsening phase stops when graph is small enough.
  2. Initial Partitioning Phase: This phase utilizes the coarsest graph given by phase one and provides a k-way partition of the graph minding the load balance. This means that this phase tries to create equal size partitions of the graph. Since the coarsest graph is usually very small this step is fast.
  3. Refinement Phase: This phase attempts to minimize communication by grouping together strongly connected nodes, preserving at the same time the load balance. The phase utilizes the k-way partition has derived from phase two. The algorithm aims to minimize the cut-set, meaning the the number of edges that cross over clusters, by moving vertices from one cluster to another. A move is considered to be useful if it reduces the cut-set and preserves the load balance.

3.6 Applications

Since the size of graph structures describing physical systems keep increasing, the question of graph partitioning becomes mandatory. This section exhibits some of the applications that use graph partitioning to reach solution:

- Parallel Processing: A classic application of graph partitioning is the distribution of load at processors of a parallel machine. Graph partitioning should ensure the balance and minimize the communication between the processors.
- Power Grids: Partitioning algorithms can be used to split a power grid into a number of self-sufficient sub-grids in order to prevent the propagation of disturbances. The objectives of the partitioning along with load shedding enhance the robustness of the power grid.
- Biological Networks: Many biological networks can be modeled using graph representation. The nodes of the graph are biological entities and the edges correspond to their common participation in some biological process. It is useful to partition these networks for many reasons, for example to reduce the data, by grouping together in a cluster nodes that behave biologically similarly to each other.
- Image Processing: Concerning image segmentation, algorithms aim to divide the pixels of an image into groups that correspond to objects in order to obtain a more compressed representation of the image referring to objects rather than pixels. Each image pixel corresponds to a node in the graph and nodes are connected by a weighted edge if they are similar to each other. The weight represents an other quantity, like the difference in the intensity between the connected pixels-nodes.
- VLSI Physical Design: Physical design of digital circuits for VLSI systems is also a very common application for graph partitioning techniques. The goal is to reduce the complexity of the system. this can be achieved by partitioning the components into smaller groups and keep the length of wires short. In graphical form the digital circuits is an assemblage of subcircuits -denoted by the clusters of the graph-, where the nodes are functional units of the circuit and the edges are the wires. The actual objective is to minimize the total communication between subcircuits. In addition, graph partitioning concerning this certain area of VLSI systems is usually associated with some constraint. For example, computation time of partitioning matters and even for circuits with million components should be small.

Chapter 4

Model Order Reduction

Speaking of Model Order Reduction we refer to a sum of techniques that aim to approach a large and complex mathematical model by reducing its order. In this chapter we will present a certain projection-based technique that uses the partition of a graph to obtain a reduced order model.

4.1 Motivation

Mathematical models are clearly created for a reason. They attempt to describe the behavior of physical systems in terms of state variables. The goal is to simulate and, eventually control the system. The model of a dynamical system consists of a number of differential equations, ordinary or partial- which are often discretized into ordinary. This number is usually large, making the model too complex for further processing. The question is if it is possible to find a simpler model, while at the same time preserve the behavior of the original one. Speaking of ”simpler” model we refer to a liner time invariant one that consists of fewer state variables than the original. Specifically, having a model of order n, model order reduction techniques are responsible to replace it with one of lower order, saying r, taking into consideration that the reduced system must behave as closely as possible equal to the original one [3],[16].

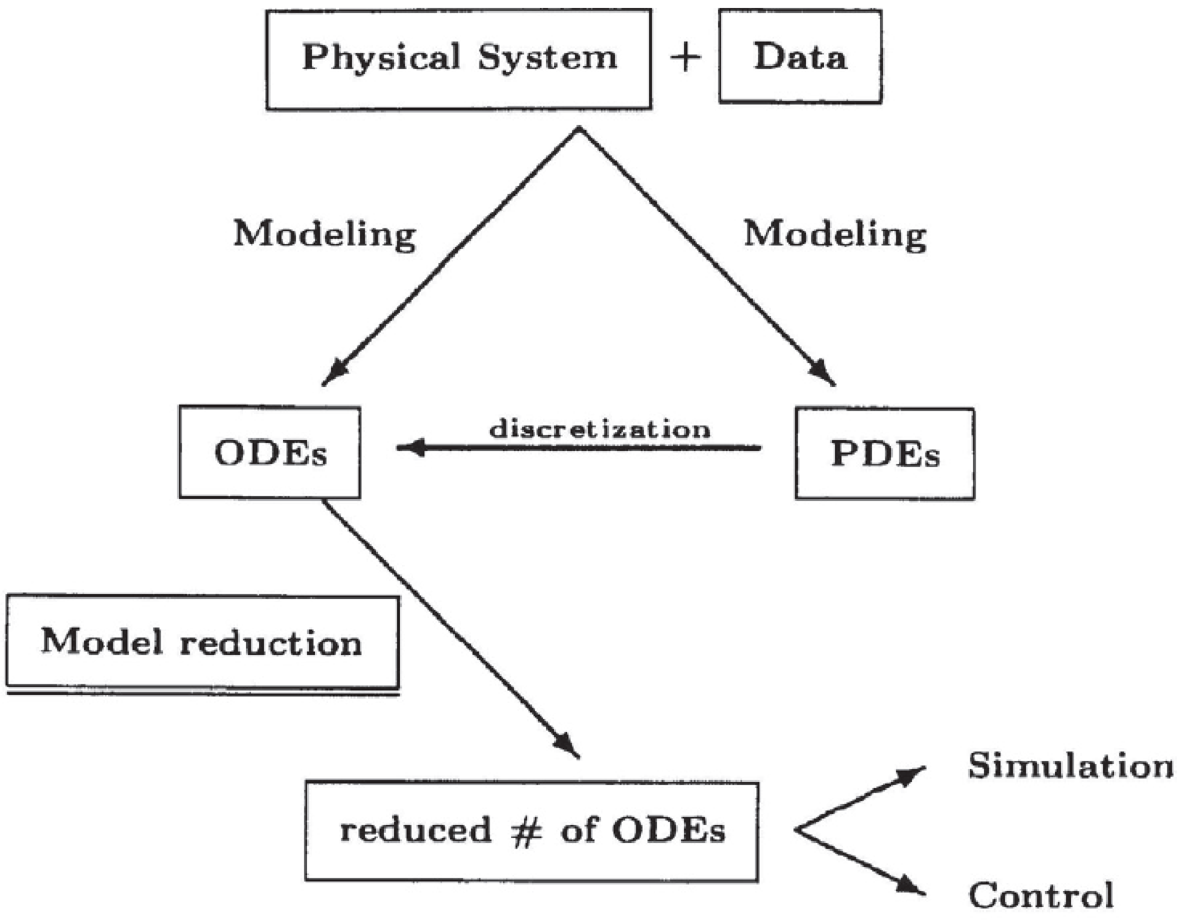


Figure 4.1: Model Order Reduction from [3]

Model order reduction is a technique having a wide range of applications. The ones described previously in graph partitioning section are between them. It is also very important at optimal control area, aiming to reduce the order of controller. A wide area of applications for order reduction techniques arises from circuit simulation problems. We need to simulate circuits consisting of million resistors, capacitors and inductors as well as active devices, like transistors and diodes. The models used for these active devices are indefinite themselves. Moreover, the more and more increasing frequencies have brought up problems concerning the electromagnetic effects in interconnection structure, that can affect the transmission of signals. To overcome this problem we need to solve the Maxwell equations resulting to very large systems.

The importance of model order reduction can be summarized [8]:

- To obtain lower-order model so as to make the understanding of the system simpler.
- To reduce computational efforts in simulation.
- To make the design numerically more efficient.

- To obtain simpler control laws.

4.2 Projection Based Model Order Reduction

MOR approaches are classified in two major categories: those based on projection techniques and the ones based on truncation. All of them aim to obtain a reduced model preserving the most useful information and removing the less important one. Their difference lies in how each technique measures importance.

Projection methods attempt to approach the high-dimensional state vector using a new vector belonging to a lower-dimension subspace. They mostly depend on the structure of the original model. In addition, they are simpler and cheaper to implement. On the other hand, truncation techniques attempt to preserve key characteristics of the system, like controllability and observability[17]. They obviously achieve better performance giving optimal model. However the basic disadvantage is that their implementation is more expensive.

4.2.1 Projection Method

Consider the eigenproblem [20]:

$$Au = lu$$

where $A \in \mathscr{R}^{n\times n}$ and $\lambda \in R$.

Assume that there is a lower-dimensional subspace M, let's say M $\mathscr{R}^{r\times r}$. A projection method, like Galerkin Projection, seeks for $\tilde{u}$,$\tilde{\lambda}$, that satisfy:

$$\tilde{u} \in M \tag{4.1a}$$

$$(A\tilde{u} - \tilde{\lambda}\tilde{u}) \perp M \tag{4.1b}$$

That means $(A\tilde{u} - \tilde{\lambda}\tilde{u})\perp M$ must be orthogonal to any base of subspace M. Let $V = \{v_1, v_2, ...v_r\}$ be an orthogonal base of M. Then if y is a vector in $\mathscr{R}^n$ equation (4.1) can be written:

$$\tilde{u} = Vy \tag{4.2a}$$

$$V^T(AVy - \tilde{\lambda}Vy) = 0 \tag{4.2b}$$

Let $P = V^T AV$ be the projection matrix. Then the original eigenproblem is written:

$$Hy = \tilde{\lambda}y \tag{4.3}$$

which is a simpler problem to solve.

4.2.2 Petrov-Galerkin Condition

Consider the model derived from modified nodal analysis that is described by (2.26),(2.30).

$$Ex'(t) + Ax(t) - Bu(t) = 0 \tag{4.4a}$$
$$y(t) = Cx(t) \tag{4.4b}$$

These equations are $\in \mathscr{R}^{n\times n}$. We aim to project them onto a lower-dimension subspace, of dinmesion r, where r«n [1], [5]. We chose subspaces V and W as well as the corresponding test basis matrices V and W, both $\in \mathscr{R}^{n\times r}$. Then the state vector x(t) is expressed in terms of $\tilde{x(t)} \in \mathscr{R}^{r}$:

$$x(t) = V\tilde{x(t)} \tag{4.5}$$

and based on that (4.4) is rewritten:

$$EV\tilde{x(t)}' + AV\tilde{x(t)} - Bu(t) = 0 \tag{4.6}$$

Applying Petrov-Galerkin condition to (4.6)

$$W^T(EV\tilde{x(t)}' + AV\tilde{x(t)} - Bu(t)) = 0 \tag{4.7}$$

leads to the reduced model:

$$\tilde{E}\tilde{x(t)}' + \tilde{A}\tilde{x(t)} - \tilde{B}u(t) = 0 \tag{4.8a}$$
$$\tilde{y} = \tilde{C}\tilde{x(t)} \tag{4.8b}$$

where the reduced matrices are given by:

$$\tilde{E} = W^T EV \tag{4.9a}$$
$$\tilde{A} = W^T AV \tag{4.9b}$$
$$\tilde{B} = W^T B \tag{4.9c}$$
$$\tilde{C} = CV \tag{4.9d}$$

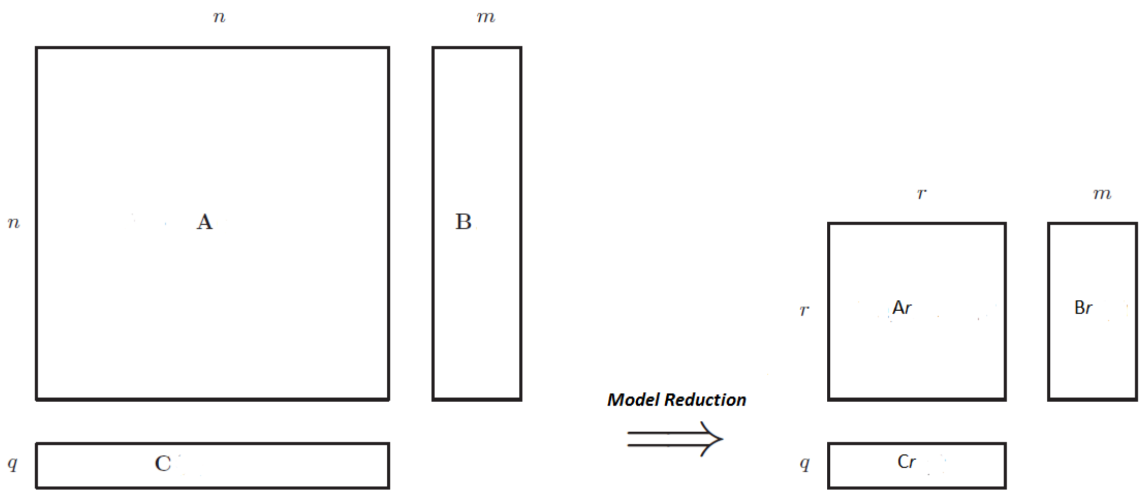


Figure 4.2: Model Order Reduction Based on Projection from [5]

Note that the equality $W^T V = I$ is true. In special case that $W = V$ the projection is orthogonal and is called just Galerkin projection.

4.3 Projection Using Graph Partitions

Let's take as an example the system represented by the graph shown in Figure 3.3. The set $V = \{v_1, v_2, ...v_n\}$ is the set of vertices and there is a subset of V, say $V_L = \{v_1, v_2, ..., v_m\}$, to which the system inputs are applied. Matrix B is responsible to assign the input vector $u = \{u_1, u_2, ..., u_m\}$ to the corresponding vertices, while matrix A includes the edge weights. Finally, column vector x denotes the sate vector of size n. The system is described by the following model:

$$x' = -Ax + Bu \tag{4.10}$$

where A is the Laplacian $n \times n$ matrix of the graph while B is $n \times m$ matrix. Supposing that the input is applied to vertices six and seven, B matrix is determined:

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.11}$$

The Laplacian matrix associated with the graph in Figure 3.3 is:

$$A = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & -5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & -3 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & -1 & -2 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 6 & -5 & -0 & 0 & 0 & 0 & 0 \\ 0 & -3 & -2 & -5 & 25 & -2 & -6 & -7 & 0 & 0 \\ -5 & -2 & -3 & 0 & -2 & 25 & -6 & -7 & 0 & 0 \\ 0 & 0 & 0 & 0 & -6 & -6 & 15 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & -7 & -7 & -1 & 15 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \tag{4.12}$$

We aim to project the model in (4.10) onto a lower-dimension subspace, using Petrov-Galerkin projection that described previously. It is significant to preserve a transparent relationship between the original system and the reduced one and a direct application of Petrov-Galerkin method may be destructive. Taking this into account a projection procedure using graph partitions to determine V and W matrices is proposed.

In previous chapter, and specifically in (3.11), we defined the partition $\pi = \{\{1,2,3,4\},\{5,6\},\{7\},\{8\},\{9,10\}\}$ of the graph G shown in Figure 3.3. Furthermore we defined the characteristic matrix P associated with that partition. Then, matrices W and V are formed based on matrix P as follows:

$$W = P(P^T P)^{-}1 \tag{4.13a}$$

$$V = P \tag{4.13b}$$

The columns of P are orthogonal, thus the matrix $(P^T P)$ is a diagonal matrix with its diagonal elements be equal to the number of vertices in each cell C_i. Hence, this matrix is invertible.

By applying Petrov-Galerkin Projection to model (4.10) we obtain the reduced one:

$$\tilde{x}' = -\tilde{A}\tilde{x} + \tilde{B}u \tag{4.14}$$

where $\tilde{x} \in \mathscr{R}^r$, $\tilde{A} = W^T A V$ and $\tilde{B} = W^T B$. Note that r is equal to the number of cells in partition π.

Considering the dimensions of W and V, $\tilde{A}$ is a $r \times r$ matrix, while $\tilde{M}$ is a $r \times m$ matrix. For this particular example r is equal to five. Matrices $\tilde{A}$ and $\tilde{B}$ corresponding to reduced model are:

$$\tilde{A} = \begin{bmatrix} 5 & -5 & 0 & 0 & 0 \\ -10 & 23 & -6 & -7 & 0 \\ 0 & -12 & 15 & -1 & -2 \\ 0 & -14 & -1 & 15 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix} \tag{4.15}$$

$$\tilde{B} = \begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.16}$$

The input weights in $\tilde{B}$ depend on the cardinality of cells in π. Originally, input u_1 is applied to vertex six, which belongs to cell C_2 and furthermore to vertex two in $\tilde{G}$. So input u_1 is now associated with $\tilde{x_2}'$ in terms of the average of input received by vertices in $C_2 = \{5, 6\}$.

The reduced graph $\tilde{G}$ is shown in Figure 3.4.

Chapter 5

Order Reduction of RC Network Model

In this chapter we regard the model derived from an RC Network as the original and we aim to obtain a reduced one applying the model order reduction technique based on graph partitions.

5.1 Problem Definition

Our original model is defined by the following equations:

$$Cx'(t) = -Gx(t) + Bu(t) \tag{5.1a}$$

$$y(t) = B^T x(t) \tag{5.1b}$$

The equations above have derived from the Modified Nodal Analysis Method. Vector x is the state vector of dimension n and vector u is the input vector of dimension m. Matrices C and G are the capatiance and conductance matrices respectively. They are both $n \times n$ sparse symmetric matrices.Moreover, matrices C and G are similar to each other, meaning that they contain elements for the same pairs of $\{i, j\}$. Matrix B is responsible for assigning the input vector to the corresponding nodes while B^T expresses output vector y in terms of state vector. Matrix B is $n \times m$. At this particular example n is equal to 288 and m equal to 32. Hence, C and G matrices are 288×288 and B is 288×32.

Firstly, we need to determine a partition π of the corresponding to matrix C graph and the characteristic matrix P associated with that. The next step requires to form projection matrices V and W based on characteristic matrix P. Then, we are ready to apply Petrov-Galerkin projection to obtain the reduced order model. Since matrices G and C are similar to each other we apply the same projection to both of them.

In order to determine the partition of the graph we are going to use alternative methods. The first is a software package named METIS [11] and the second one is an algorithm fully implemented with Matlab commands [10].

5.2 Partitioning Using METIS Software Package

5.2.1 METIS Overview

METIS is a serial software package for partitioning large irregular graphs, partitioning large meshes, and computing fill-reducing orderings of sparse matrices. METIS has been developed at the Department of Computer Science & Engineering at the University of Minnesota and is freely distributed. The algorithms implemented in METIS are based on the multilevel graph partitioning approach, and they seem produce high-quality partitionings and fill-reducing orderings in relatively small amount of time fill-reducing orderings.

METIS can provide a k-partition of a given graph, where k is specified by user, using either multilevel recursive bisection or multilevel k-way partitioning algorithms. Both methods provide high quality partitions, however multilevel k-way partition algorithm is preferable, cause it provides additional capabilities.METIS stand-alone program for partitioning is gpmetis.

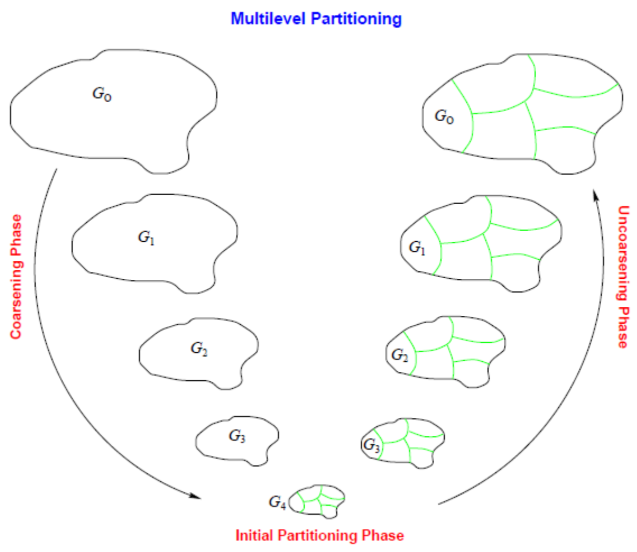


Figure 5.1: The three phases of multilevel k-way graph partitioning. During the coarsening phase, the size of the graph is successively decreased. During the initial partitioning phase, a k-way partitioning is computed, During the multilevel refinement (or uncoarsening) phase, the partitioning is successively refined as it is projected to the larger graphs. G0 is the input graph, which is the finest graph. G_{i+1} is the next level coarser graph of G_i. G_4 is the coarsest graph. Figure from [11]

The objective of graph partitioning problem is to obtain a k-way partitioning such that the number of edges, or in case of weighted graphs the sum of weights of edges, that cross different clusters of partition is minimized. This objective is commonly called the edge-cut.

5.2.2 METIS API Routine

All routines implemented in METIS can be called from C/C_{++} programs(Appendix A). The routine we are going to use is k-way graph partitioning routine which is defined as follows:

```
  int METIS_PartGraphKway(idx_t *nvtxs, idx_t *ncon, idx_t *xadj,
             idx_t *adjncy,idx_t *vwgt, idx_t *vsize, idx_t *adjwgt,
             idx_t *nparts, real_t *tpwgts, real_t ubvec,
             idx_t *options,idx_t *objval, idx_t *part)
```

Figure 5.2: METIS Routine from [11]

$idx_$t is a special data type for storing integer quantities used by METIS and it is defined in metis.h library. There is a $real_t$ data type for storing floating point quantities, also defined in metis.h. All routines in METIS take inputs that are of these two data types. Moreover, METIS allows some parameters to have NULL value in order to minimize the complexity.

The list of parameters of the routine is described in the next table. The parameters allowed to be NULL are indicated with a (NULL) following them.

Parameter	Description
nvtxs	The number of vertices in the graph
ncon	The number of balancing constraints. It should be at least 1
xadj, adjncy	The adjacency structure of the graph as described in next lines
vwgt (NULL)	The weights of the vertices
vsize (NULL)	The size of the vertices for computing the total communication volume
adjwgt (NULL)	The weights of the edges
nparts	The number of parts to partition the graph
tpwgts (NULL)	This is an array of size $nparts \times ncon$ that specifies the desired weight for each partition and constraint. A NULL value can be passed to indicate that the graph should be equally divided among the partitions
ubvec (NULL)	This is an array of size ncon that specifies the allowed load imbalance tolerance for each constraint. A NULL value can be passed indicating that the load imbalance tolerance for each constraint should be 1.001 (for ncon=1)
options (NULL)	This is the array of options
objval	Upon successful completion, this variable stores the edge-cut of the partitioning
part	This is a vector of size nvtxs that upon successful completion stores the partition vector of the graph

Table 5.1: List of Routine's Parameters

In order to include a graph in the parameter list of the routine we need its adjacency structure, using compressed storage format(CSR). In this format a graph consisting of n nodes and m edges is represented by two arrays: xadj and adjncy. For each vertex i the adjncy array contains the adjacent to this vertices and it is of size 2m. Assuming that vertex numbering starts from 0, xadj[i] indicates the starting point of vertex i in adjncy matrix, while xadj[i+1] indicates, but not includes, the ending point. Array Xadj is of size n+1.

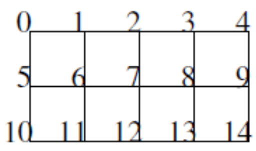


Figure 5.3: A simple graph from [11]

Then the two arrays concerning its adjacency structure are:

```
xadj 0 2 5 8 11 13 16 20 24 28 31 33 36 39 42 44
adjncy 1 5 0 2 6 1 3 7 2 4 8 3 9 0 6 10 1 5 7 11 2 6 8 12 3 7 9 13 4 8 14 5 11 6
    10 12 7 11 13 8 12 14 9 13
```

Figure 5.4: Adjancy Structure of the Simple Graph in Figure 5.3 from [11]

Note that according to xadj array for vertex 0 the adjacent vertices start from adjncy[0] and end, but not include, at element adjncy[2].

Returning back to our problem, we need to convert C matrix to adjacency form in order to be included in the parameter list of the routine. We use the following Matlab script to extract adjncy and xadj arrays. Note that we need those arrays to begin element numbering from 0 (C style), while Matlab starts element numbering from one.

```
     adjncy=zeros(1,1234)  % adjncy is of size 2*m=1234
     xadj=zeros(1,289)     % xadj is of size n+1
     xadj(1)=0;            % initialization of xadj array
     k=1;                  % pointer runs adjncy array
     m=2;                  % pointer runs xadj array

    for i=1:n
       for j=1:n
           if C(i,j) ~= 0
               adjncy(k)=j-1;
               k=k+1;
           end
       end
       xadj(m)=k-1;
       m=m+1;
   end
```

Figure 5.5: Matlab script adjacency.m

5.3 Partitioning Using grPartition Algorithm

5.3.1 Algorithm Overview

It is a graph partitioning algorithm based on spectral factorization. The algorithm is fully implemented with Matlab commands (appendix B). The objective is to find a k-partition that minimizes the cost C(P) which is:

$$C(P) = \sum_{i \neq j} \sum_{(u,\tilde{u}) \in E / u \in V_i, \tilde{u} \in V_j} \tilde{c}(u,\tilde{u}) \tag{5.2}$$

The algorithm is called by the function:

```
% C: is the Laplacian matrix of the model
% r: is the desired number of clusters
% nrep: number of repetion for the clustering algorithm ,optional input-default
   is one
%ndex: r vector with with the cluster index for every node (indicates from 1)
%Pi: Projection matrix
%cost: cost of the partition (sum of broken edges)

[ndx,Pi,cost]= grPartition(C,r,nrep);  %nrep=1
```

Figure 5.6: Function grPartition.m

5.4 Characteristic and Projection Matrices Formulation

METIS routine is called from a C_{++} program (Appendix A), where user defines nparts parameter to be equal to the desired number of clusters. User also includes the adjncy and xadj arrays, which store the adjncy structure of the graph and specifies the number of vertices through nvtxs variable. The rest of the parameters in parameter list are set to their default values. Upon successful completion, part vector indicates at which individual cluster each vertex belongs. This is all the information needed to form the $n \times r$ characteristic matrix P and then projection matrices W and V. The routine also returns the edge-cut through objval variable.

```
R=dlmread('part.txt');      % part.txt contains part vector
n=288;                      % original number of vertices
r=specified_by_user;        % r is the desired number of clusters
for i=1:n
  R(i,1)=R(i,1)+1;          % Matlab starts element numbering from 1
end
P=zeros(n,r);               % initialize characteristic matrix P
for i=1:n
  j=M(i,1);
  P(i,j)=1;                 %characteristic matrix formulation
end
W=P*inv((P')*P);            % projection matrix W formulation
V=P;                        % projection matrix W formulation
```

Figure 5.7: Matlab script matrices_formulation.m

Note that if we use the grPartition algorithm ndx vector is used instead of part vector.

Then we form matrices $\tilde{C}$, $\tilde{G}$ and $\tilde{B}$ corresponding to the reduced model:

```
    C_red=W'*C*V;
    G_red=W'*G*V;
    B_red=W'B;
    D_red=B^T*V;
```

Figure 5.8: Matlab script reduced_model_matrices_formulation.m

Chapter 6

Numerical Results

Following we will exhibit the comparison between the original and reduced model for a certain number of clusters, but first we need to determine this number. Generally, we could partition the graph in any number r<n of clusters. However, a good aspect is to take into account the balance constraint that states [2]:

$$|V_i| \leq (1+e)\frac{n}{r} \tag{6.1}$$

where $|V_i|$ is the number of vertices in cluster C_i, n is the total number of vertices and r is the desired number of clusters. When $e = 0$, the partition is considered to be perfectly balanced. So, to begin with, we chose r based on that, for instance $r = 24$.

First, we use the following Matlab script that loads the matrices C and G and creates input matrix B and also input vector u. Furthermore, the script determines the timestep of the simulation and the number of time instants. The last two quantities are needed for the simulation in time domain.

```
     load G.dat;
     load C.dat;
     load f_io.txt;

     n = 288;

     G = sparse( G(:,1), G(:,2), G(:,3), n, n );
     C = sparse( C(:,1), C(:,2), C(:,3), n, n );

     N = length(f_io);

     p = N/2;
     q = p;

     B = zeros(n, p);

    for in_i = 1:p
      B(f_io(2*in_i-1), in_i) = 1;
    end

% Stimulus
[t, dt, e] = excitation_sel(1);
%excitation_sel in appendix C

u = zeros(length(e), N/2);

for in_i = 1:p
   u(:, in_i) = e';
end

timestamps = length(e);
```

Figure 6.1: Matlab script load.m

6.1 Comparison Between Reduced and Original Model

6.1.1 Frequency Domain

In order to compare the reduced model with the original one in frequency domain we are going to use the transfer function of the system which was described in Chapter 2. The original model is written in frequency domain for $x(0) = 0$:

$$\mathscr{L}\{Cx' = -Gx + Bu\} \Longrightarrow CX(s) = -GX(s) = BU(s) \Longrightarrow X(s) = (sC+G)^{-1} + BU(s) \tag{6.2}$$

while (5.1b) is written:

$$\mathscr{L}\{y(t) = B^T x(t)\} \Longrightarrow Y(s) = B^T U(s) \tag{6.3}$$

Hence, transfer function is defined:

$$H(s) = Y(s)/U(s) = B^T(sC+G)^{-1}B \tag{6.4}$$

The following Matlab script computes H(s) concerning the original and the reduced system, for $s = j\omega$, where $\omega \in [10^2, 10^6]$ Hz. Transfer matrix, H(s), is a $m \times m$, here 32×32 and we chose to plot some of the diagonal elements. The plot chart is common for the original and reduced model.

```
m=logspace(2,6,100);
%divides frequency range into 100 logarithmically spaced points
for i=1:100
     s=j*m(i);
     H=(abs((B')*inv((s*C+G))*B));
     H_red=(abs((B_red')*inv((s*C_red+G_red))*B_red));
     H_i(:,i)=mag2db(abs(H(1,1)));
     %computes a row vector of H(1,1) for 100 different frequencies. Magnitude
         is expressed in dbs
     H_red_i(:,i)= mag2db(abs(H_red(1,1)));
end
figure('Name','Reduced and Original Model Comparison');
plot(m, H_red_i,'r');
hold on;
plot(m, H_i,'b');
```

Figure 6.2: Matlab script comparison_frequency_domain.m

6.1.1.1 Using METIS

The above script produces the chart:

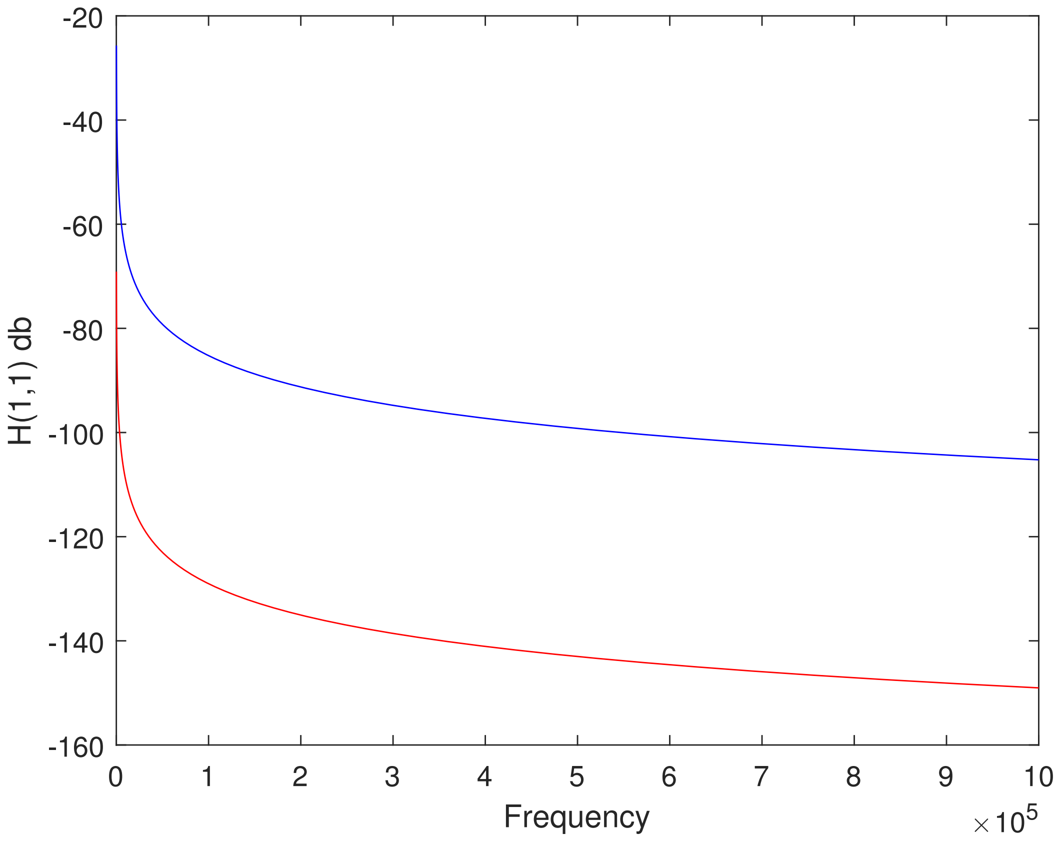
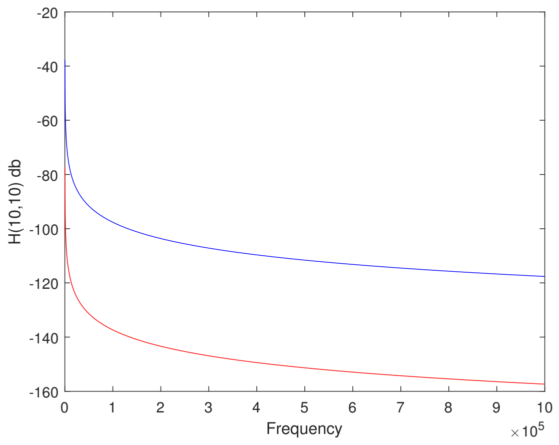
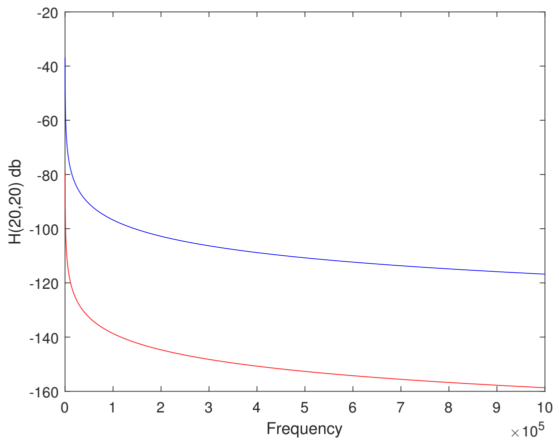


Figure 6.3: H(1,1) for reduced ('r') and original ('b') model

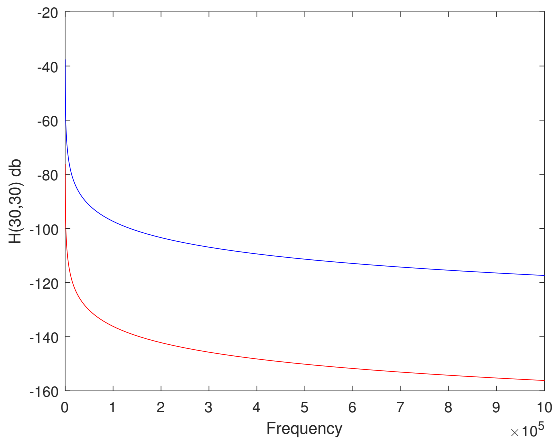
We chose to also plot H(10,10), H(20,20), H(30,30):



(a) H(10,10)



(b) H(20,20)

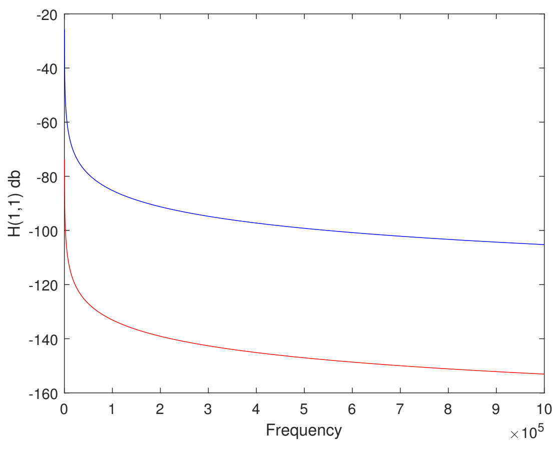


(c) H(30,30)

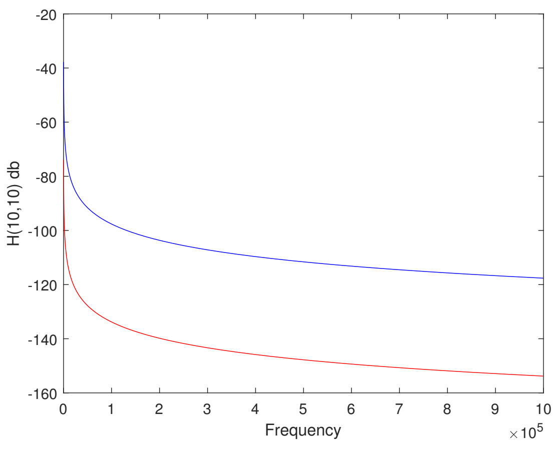
Figure 6.4: Plots of diagonal pairs of H(s)

6.1.1.2 Using grPartition Algorithm

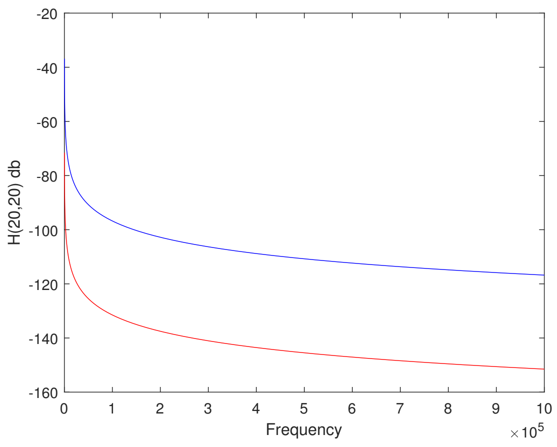
If we use grPartition algorithm to obtain a partition of the graph corresponding to matrix C the output charts are:



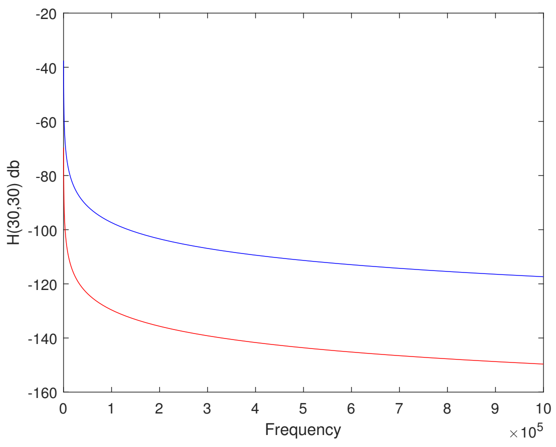
(a) H(1,1)



(b) H(10,10)



(c) H(20,20)



(d) H(30,30)

Figure 6.5: Plots of diagonal pairs of H(s)

6.1.2 Time Domain

In time domain, we attempt to approximate the output vector y, using the Implicit or Backward Euler method. The approximations are given by:

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}) \tag{6.5}$$

where h is called step.

The output vector y is given by (5.1b) and it is a column vector of size m, where $m = 32$. The following function approaches y_i computing its value for a number of time instants, specifically this number is equal to 36. So, the output y is a 32×36 matrix.

```
  % dt is the timestep of simulation,$dt = t_i - t_{i-1}$
  % x0 is a zero vector
  % timestamps is the total number of time instants in the simulation
  %D=B^T, the matrix weighting output vector

function [y] = tr_sim_be(C, G, B, D, e, dt, x0, timestamps)
   S = G + (1/dt)*C;
   [L,U] = lu(S);clc
   y = zeros(size(D,2), timestamps);
   x = x0;
   tic;
   for k = 1:32
      b = (B*e(k, :)') + (1/dt)*C*x;
      c = L\b;
      x = U\c;
      y(:, k) = D(:, k)'*x;
   end
   toc;
end
```

Figure 6.6: Matlab function sim.m

The function is called from the next script which is also responsible for plotting in common chart some certain y_i for the original and reduced model. The script plots the first output y(1) for all time instants. In addition, it plots one of the system inputs, specifically the input applied to vertex one of the original graph. Note that all inputs are of same form.

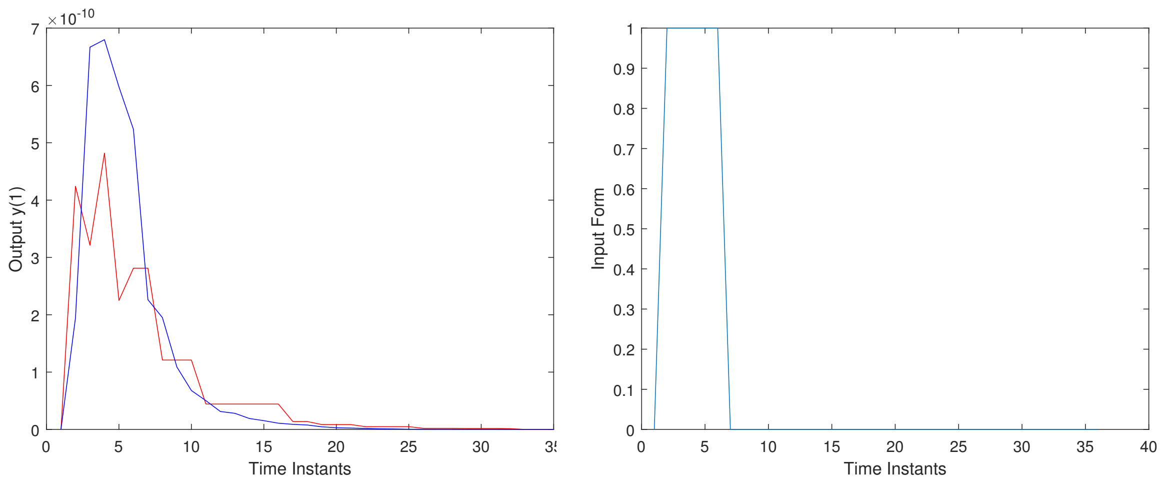
```
x0_red=zeros(r,1); %r is the dimension of reduced model vector
x0=(288,1);

y_red=sim(C_red, G_red, B_red, D_red', e, dt, x0, timestamps);
y=sim(C, G, B, B, e, dt, x0_288, timestamps);
%matrix B is used for parameter D


figure('Name','Reduced and original model comparison in time domain');
plot(y_red(1,:),'r');
hold on;
plot(y(1,:),'b');

%plot one system input
figure('Name','Input 1');
plot(u(:,1));
```

Figure 6.7: Matlab script time_domain.m

Considering the edge cut for this certain partition of 24 clusters, METIS routine returned the objval variable equal to 289, while grPartition algorithm computed the cost equal to 120.

6.1.2.1 Using METIS

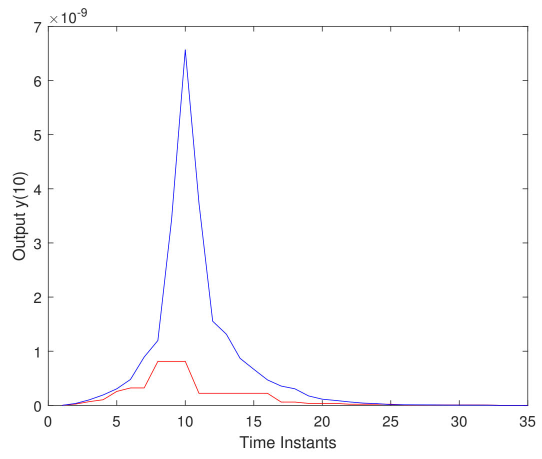
The script produces the chart:



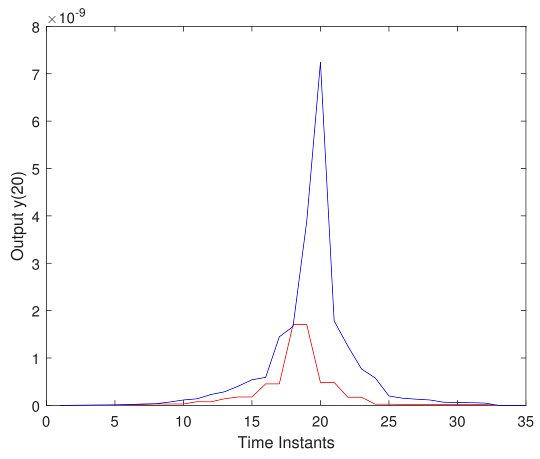
(a) y(1) for reduced ('r) and original ('b') model (b) Input 1

Figure 6.8: Script Output

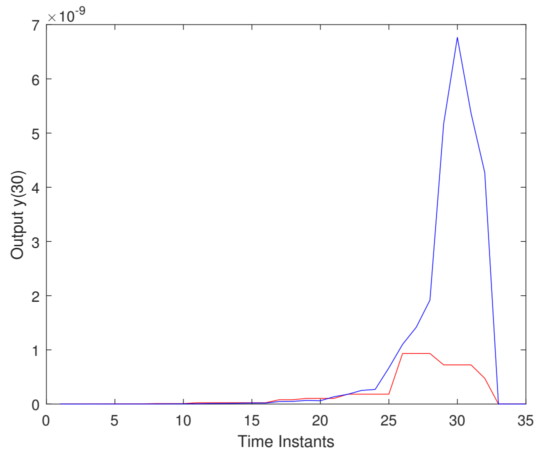
Some more outputs $y(i)$:



(a) y(10)



(b) y(20)

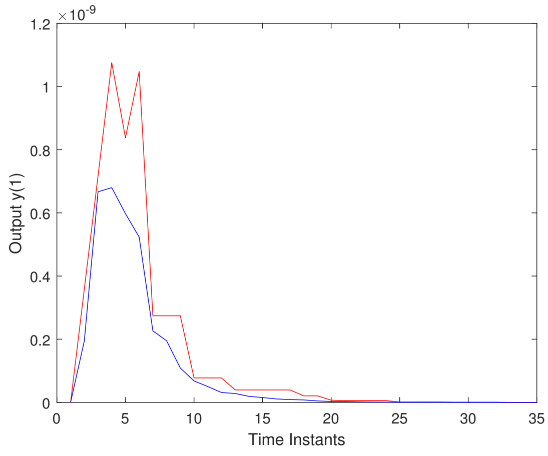


(c) y(30)

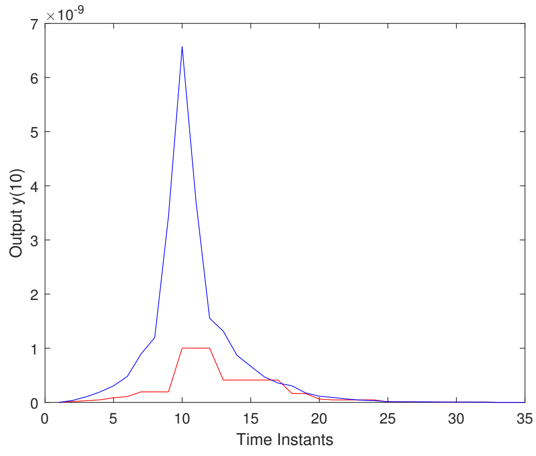
Figure 6.9: Plots of output vectors elements y(i)

6.1.2.2 Using grPartition Algorithm

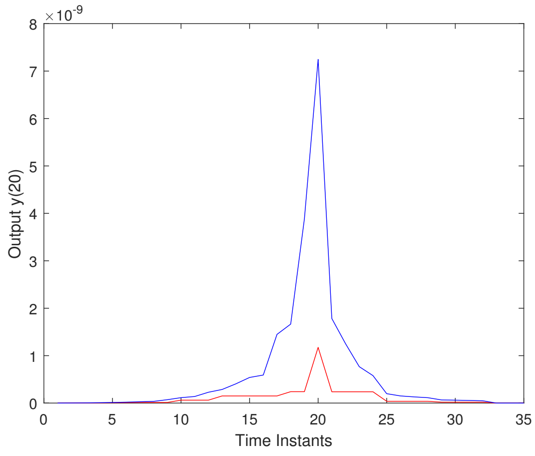
If we use grPartition algorithm the corresponding output charts are:



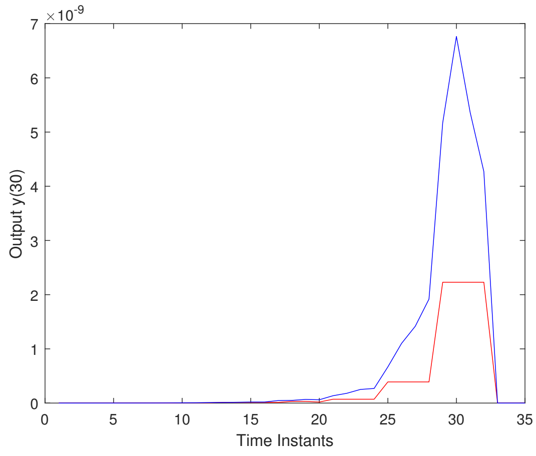
(a) y(1)



(b) y(10)



(c) y(20)



(d) y(30)

Figure 6.10: Plots of output vectors elements y(i)

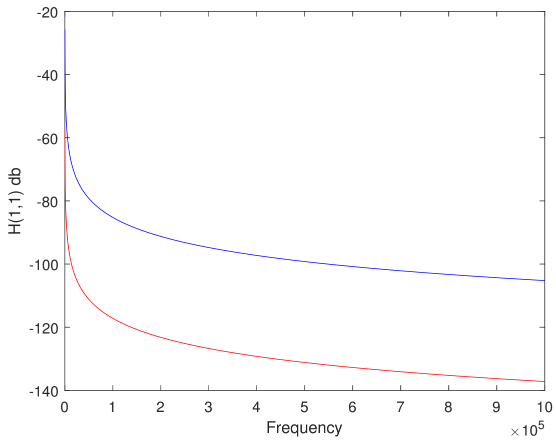
6.2 Determining a Different Partition

In previous section we presented the comparison between the original and reduced model graphically, having determined the number of clusters to be equal to 24. Now, taking into account the same balance constraint in (6.1) we define a partition that consists of 48 clusters, meaning that each cluster now will contain less vertices. The procedure and Matlab scripts used are exactly the same. The only difference is that part and ndx vector have changed to divide vertices into 48 clusters instead of 24.

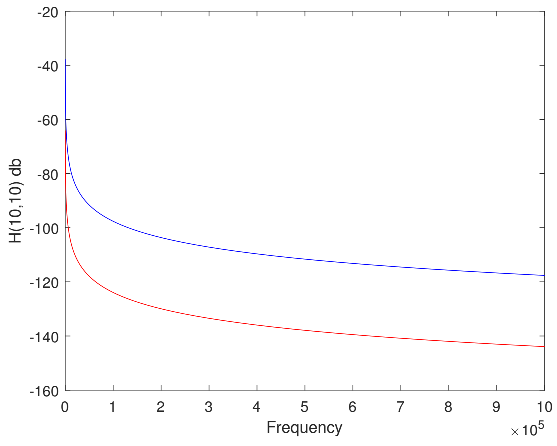
Concerning the edge cut for this partition, METIS routine determines that it is equal to 257, while grPartition algorithm returns cost variable equal to 170. Based on that we can conclude that for METIS the partition of 48 clusters is cheaper, but this is not true for grPartition algorithm.

6.2.1 Frequency Domain

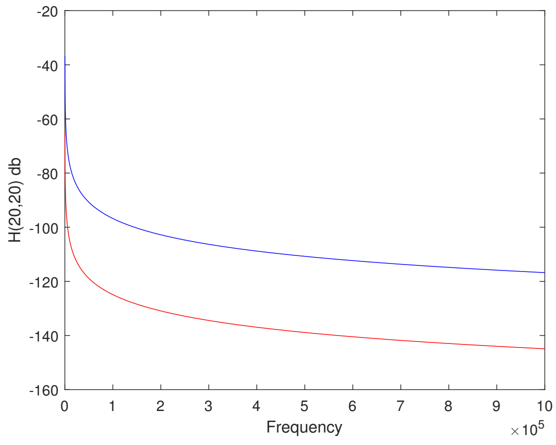
6.2.1.1 Using METIS



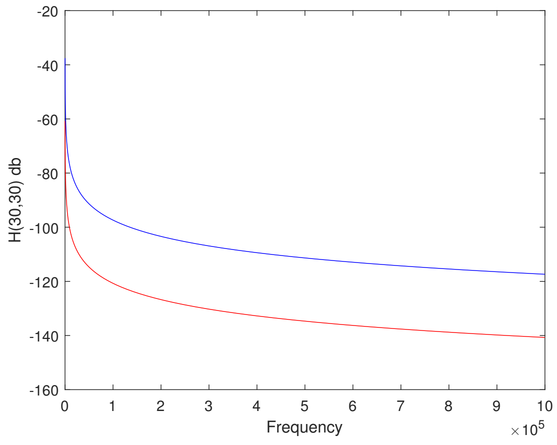
(a) H(1,1)



(b) H(10,10)



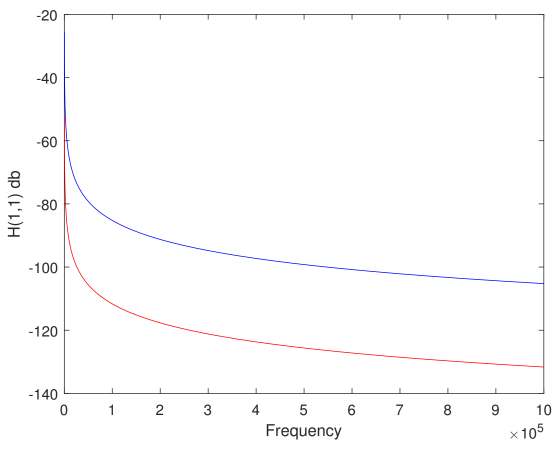
(c) H(20,20)



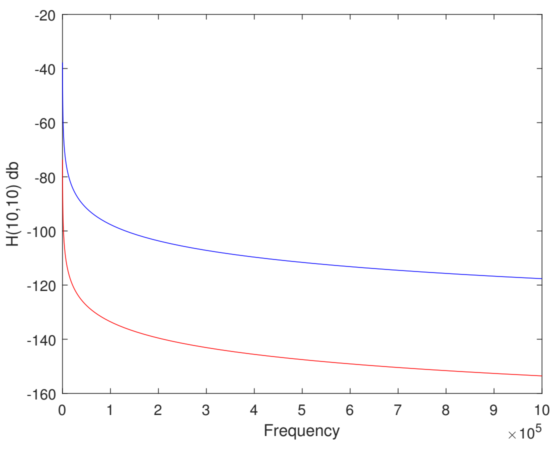
(d) H(30,30)

Figure 6.11: Plots of diagonal pairs of H(s)

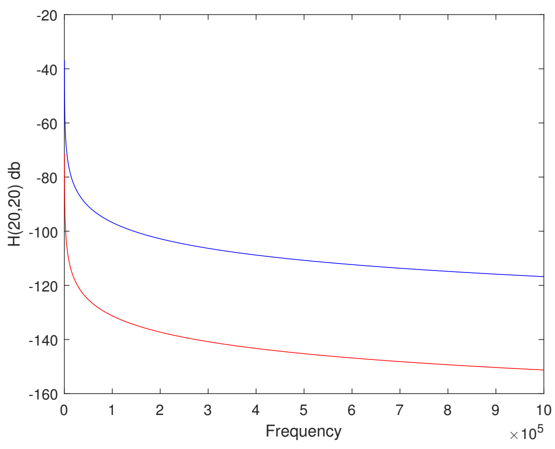
6.2.1.2 Using grPartition Algorithm



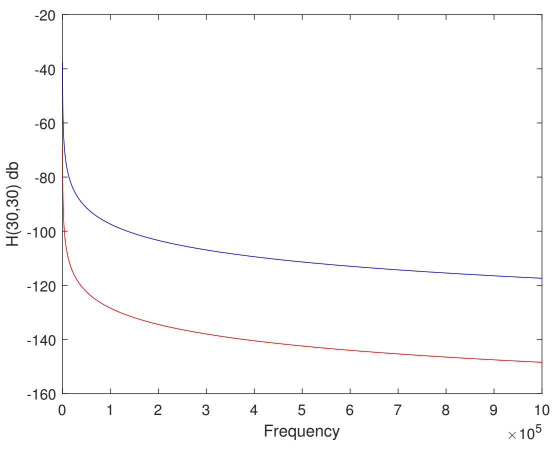
(a) H(1,1)



(b) H(10,10)



(c) H(20,20)

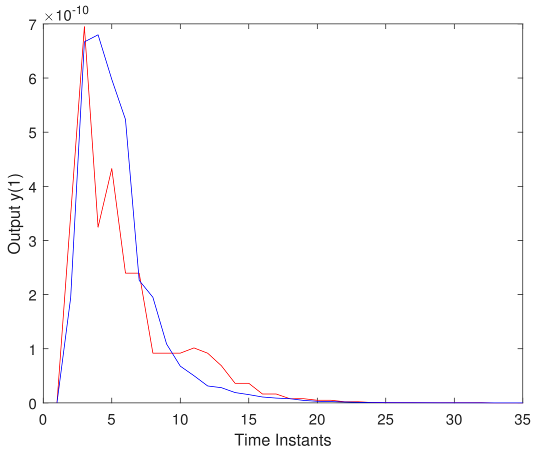


(d) H(30,30)

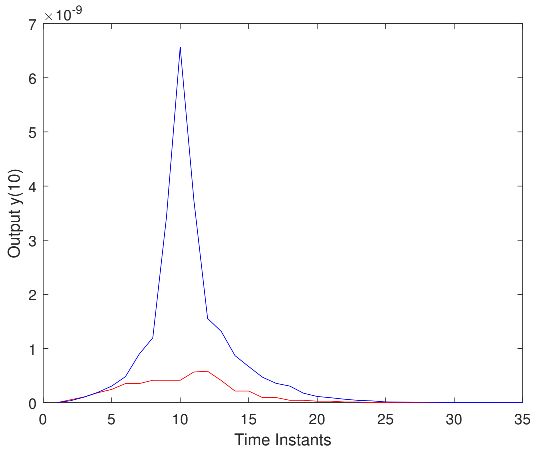
Figure 6.12: Plots of diagonal pairs of H(s)

6.2.2 Time Domain

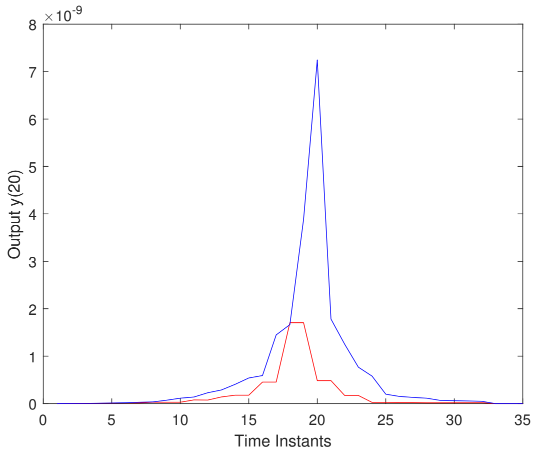
6.2.2.1 Using METIS



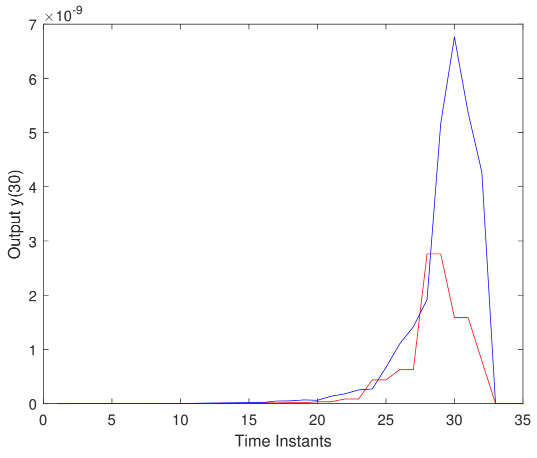
(a) y(1)



(b) y(10)



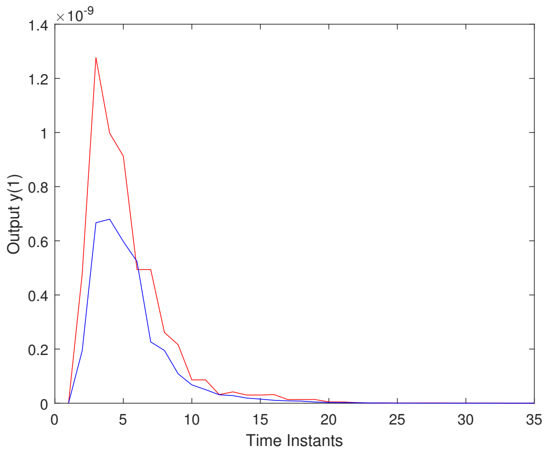
(c) y(20)



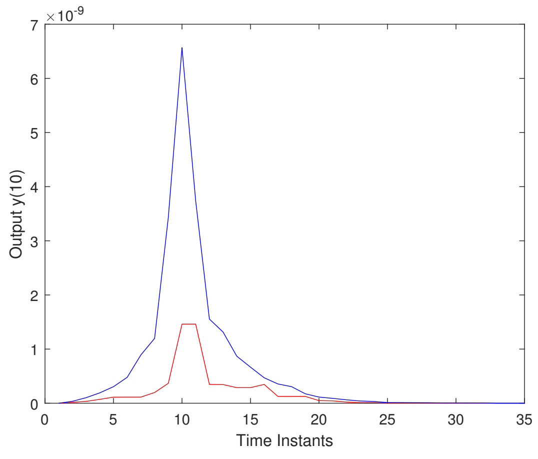
(d) y(30)

Figure 6.13: Plots of output vector elements y(i)

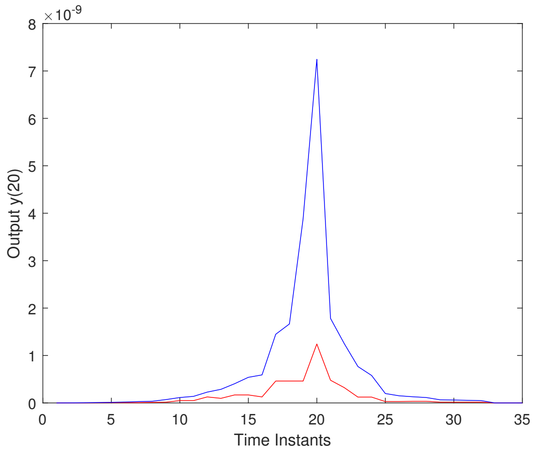
6.2.2.2 Using grPartition Algorithm



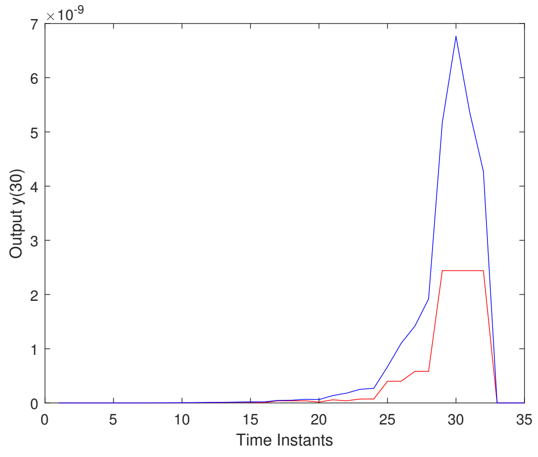
(a) y(1)



(b) y(10)



(c) y(20)



(d) y(30)

Figure 6.14: Plots of output vector elements y(i)

Chapter 7

Conclusion

7.1 Summary

In this thesis, a certain technique to achieve order reduction of high order models was described. The technique is based on projection, meaning that it reduces the high dimension of a state vector x by mapping it into a new state vector of lower dimension. We regarded the Laplacian matrix of the model as a directed and weighted graph and we used a partition of it to determine projection matrices. After applying projection we obtained a reduced graph having fewer vertices, which correspond to fewer state variables. The reduced graph is again associated with a new, also reduced Laplacian matrix.

In the last chapter numerical results were presented. We used Matlab software to compare the behavior of the original and reduced model in frequency and time domain, using two alternative methods to partition the graph. We observed that the reduced model approaches the behavior of the original one. We repeated the procedure using a different partition of the graph, consisting of smaller clusters, and we observed that behaves even more similarly to the original one.

Generally, model order reduction is a crucial technique, of grate importance in the field of approaching large and complex physical systems. It allows the extraction of simpler mathematical models, so we are eventually able to simulate, control or optimize the original complex systems of interest.

7.2 Further Work

In this thesis we described one effective technique to achieve the reduction of a model that has derived from a linear time invariant system. Clearly, this one piece of research concerning model order reduction concept. There are further challenging problems, like the reduction of nonlinear or time varying systems.

Appendix A

Appendix A

The current appendix exhibits the C++ program used to call METIS routine. METIS can be downloaded from // http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/metis-5.1.0.tar.gz for free.

```
    #include <cstddef> /* NULL */
    #include <metis.h>
    #include <iostream>
    #include <stdio.h>

int main(){

    idx_t nVertices = 288;
    idx_t nEdges  = 617;
    idx_t nWeights = 1;
    idx_t nParts  = 24;
    idx_t objval;
    idx_t part[nVertices];

    // Indexes of starting points in adjacent array
    idx_t xadj[nVertices+1] =
        {0,2,6,11,15,20,24,...1216,1220,1224,1228,1230,1232,1234};


    // Adjacent vertices in consecutive index order
    idx_t adjncy[2 * nEdges] =
        {0,1,0,1,2,3,1,2,4,16,17,1,3,4,5,2,3,4,6,18,...,285,278,286,284,287};



     int ret = METIS_PartGraphKway(&nVertices,& nWeights, xadj, adjncy,
                   NULL, NULL, NULL, &nParts, NULL,
                   NULL, NULL, &objval, part);

     std::cout << ret << std::endl;
     for(unsigned part_i = 0; part_i < nVertices; part_i++){
     std::cout << part_i << " " << part[part_i] << std::endl;
     }
     std::cout << objval << std::endl;
    return 0;
}
```

Figure A.1: C++ program to call METIS routine

Appendix B

Appendix B

This appendix exhibits the grPartion Algorithm implementation in Matlab [10].

```
       function [ndx,Pi,cost]= grPartition(C,k,nrep);
%
% function [ndx,Pi,cost]= grPartition(C,k,nrep);
%
% Partitions the n-node undirected graph G defined by the matrix C
%
% Inputs:
% C - n by n edge-weights matrix. In particular, c(i,j)=c(j,i) is equal
%    to the cost associated with cuting the edge between nodes i and j.
%    This matrix should be symmetric and doubly stochastic. If this
%    is not the case, this matrix will be normalized to
%    satisfy these properties (with a warning).
% k - desired number of partitions
% nrep - number of repetion for the clustering algorithm
%      (optional input, defaults to 1)
%
% Outputs:
% ndx - n-vector with the cluster index for every node
%      (indices from 1 to k)
% Pi  - Projection matrix [see Technical report
% cost - cost of the partition (sum of broken edges)
%
% Example:
%
% X=rand(200,2);            % place random points in the plane
% C=pdist(X,'euclidean');  % compute distance between points
% C=exp(-.1*squareform(C)); % edge cost is a negative exponential of distance
%
% k=6;                      % # of partitions
% [ndx,Pi,cost]= grPartition(C,k,30);
%
% colors=hsv(k);            % plots points with appropriate colors
% colormap(colors)
% cla
% line(X(:,1),X(:,2),'MarkerEdgeColor',[0,0,0],'linestyle','none','marker','.');
% for i=1:k
%   line(X(find(ndx==i),1),X(find(ndx==i),2),...
%      'MarkerEdgeColor',colors(i,:),'linestyle','none','marker','.');
% end
% title(sprintf('Cost %g',cost))
% colorbar
%
% Copyright (c) 2004, Joao Hespanha
% All rights reserved.

% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions
% are met:
%   * Redistributions of source code must retain the above copyright
%   notice, this list of conditions and the following disclaimer.
```

```
    %   * Redistributions in binary form must reproduce the above
%   copyright notice, this list of conditions and the following
%   disclaimer in the documentation and/or other materials provided
%   with the distribution.
%
%   * Neither the name of the <ORGANIZATION> nor the names of its
%   contributors may be used to endorse or promote products derived
%   from this software without specific prior written permission.
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
% "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
% LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
% FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
% COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
% INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
% BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
% LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
% CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
% LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
% ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
%
% Modification history:
%
% October 21, 2006
% Use of stats/kmeans can be bypassed is the Statistics toolbox is not
% available, by using mykmeans.m
%
% Augeust 27, 2006
% GNU GPL added
%
% Before August 27, 2006
% No idea about what happened before this date...

if nargin<3
  nrep=1;
end

[n,m]=size(C);
if n~=m
  error('grPartition: Cost matrix is not square');
end

if ~issparse(C)
  C=sparse(C);
end
```

```
   % Test for symmetry
if any(any(C~=C'))
  warning('grPartition: Cost matrix not symmetric, making it symmetric')
  % Make C symmetric
  C=(C+C')/2;
end

% Test for double stochasticity
if any(sum(C,1)~=1)
  warning('grPartition: Cost matrix not doubly stochastic, normalizing
      it.','grPartition:not doubly stochastic')
  % Make C double stochastic
  C=C/(1.001*max(sum(C))); % make largest sum a little smaller
                           % than 1 to make sure no entry of C becomes negative
  C=C+sparse(1:n,1:n,1-sum(C));
  if any(C(:))<0
    error('grPartition: Normalization resulted in negative costs. BUG.')
  end
end

if any(any(C<0))
   C=abs(C);

end

% Spectral partition
options.issym=1;          % matrix is symmetric
options.isreal=1;         % matrix is real
options.tol=1e-6;         % decrease tolerance
options.maxit=500;        % increase maximum number of iterations
options.disp=0;
[U,D]=eigs(C,k,'la',options); % only compute 'k' largest eigenvalues/vectors


if exist('kmeans')==2
   % Clustering -- REQUIRES the Statistics Toolbox
   %ndx=kmeans(U,k,'Distance','cosine','Start','sample','Replicates',nrep,'EmptyAction','singlet
   ndx=kmeans(U,k,'Distance','cosine','Start','sample','Replicates',nrep);
elseif exist('litekmeans')==2
   % requires litekmeans from file exchange
   ndx=litekmeans(U',k);
   %ndx=kmeans(U,k);
else
   %  warning('grPartition: Statistics Toolbox not available, using
      mykmeans.m','grPartition:no Statistics Toolbox')
   %  Simple case
   %  [Q,R,E]=qr(U',0);
   %  Z=Q*(R(:,1:k)')^(-1);
   %  [dummy,ndx]=max(U*Z,[],2);
   ndx=mykmeans(U,k,100,nrep);
end
```

```
if nargout>1
Pi=sparse(1:length(ndx),ndx,1);
end

if nargout>2
    cost=full(sum(sum(C))-trace(Pi'*C*Pi));
end

return
```

Figure B.1: grPartition algorithm implementation in Matlab

Appendix C

Appendix C

This appendix exhibits the matlab script excitation_sel.m used in Chapter 6.

```
    function [t, dt, e] = excitation_sel(sel)

   dt = 1e-9;
   t = (0:dt:35000e-12)'; % zoom into the transient phenomenon

   e = zeros(length(t), 1);

   if (sel == 1)
       ref_t = 1*dt;
       width = 5;
       unitstep = and(t >= ref_t, t <= ref_t + width*dt);
       e = unitstep;
   elseif (sel == 2)
       ref_t = 1*dt;
       rise_t = 5*dt;
       unitstep = and(t >= ref_t, t <= ref_t+rise_t);
       ramp = t.*unitstep;
       ramp = ramp(ref_t/dt + 1:ref_t/dt + 1 + rise_t/dt) - ref_t;
       e = [zeros(ref_t/dt, 1); ramp; ramp(length(ramp)).*ones(length(t) -
           length(ramp) - ref_t/dt, 1)];
   else
       ref_t = 5*dt;
       delta = (t == ref_t);
       e = delta;
   end

end
```

Figure C.1: excitation_sel.m

Bibliography

[1] David Amsallem and Charbel Farhat.
Stabilization of projection-based reduced-order models.
International Journal for Numerical Methods in Engineering, 91(4):358–377, 2012.

[2] Konstantin Andreev and Harald Racke.
Balanced graph partitioning.
Theory of Computing Systems, 39(6):929–939, 2006.

[3] Athanasios C Antoulas.
An overview of approximation methods for large-scale dynamical systems.
Annual reviews in Control, 29(2):181–190, 2005.

[4] Athanasios C Antoulas, Roxana Ionutiu, Nelson Martins, E Jan W ter Maten, Kasra Mohaghegh, Roland Pulch, Joost Rommes, Maryam Saadvandi, and Michael Striebel.
Model order reduction: methods, concepts and properties.
In *Coupled multiscale simulation and optimization in nanoelectronics*, pages 159–265. Springer, 2015.

[5] Peter Benner, Serkan Gugercin, and Karen Willcox.
A survey of projection-based model reduction methods for parametric dynamical systems.
SIAM review, 57(4):483–531, 2015.

[6] Peter Benner, Michael Hinze, and E Jan W Ter Maten.
Model reduction for circuit simulation.
Springer, 2011.

[7] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz.
Recent advances in graph partitioning.
In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[8] Luigi Fortuna, Giuseppe Nunnari, and Antonio Gallo.
Model order reduction techniques with applications in electrical engineering.
Springer Science & Business Media, 1992.

[9] Trevor M. Fowler.
LTI mathematical fundamentals.
In *LTI System and Control Theory*, pages 3–32, 2007.

[10] Joao P Hespanha.
An efficient Matlab algorithm for graph partitioning.
Santa Barbara, CA, USA: University of California, 2004.

[11] George Karypis.
METIS, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices,version 5.1.0,http://www.cs.umn.edu/˜karypis.
2013.

[12] Mehran Mesbahi and Magnus Egerstedt.
Graph theory.
In *Graph Theoretic Methods in Multiagent Networks*, pages 14–26, 2010.

[13] Nima Monshizadeh, Harry L Trentelman, and M Kanat Camlibel.
Projection-based model reduction of multi-agent systems using graph partitions.
IEEE Transactions on Control of Network Systems, 1(2):145–154, 2014.

[14] Santosh Nage and Girish Potdar.
A survey on graph partitioning techniques.
In *International Journal of Science and Research (IJSR)*. 2013.

[15] Derek Rowell.
State-space representation of LTI systems.
URL: http://web. mit. edu/2.14/www/Handouts/StateSpace. pdf, 2002.

[16] Wilhelmus HA Schilders, Henk A Van der Vorst, and Joost Rommes.
Model order reduction: theory, research aspects and applications, volume 13.
Springer, 2008.

[17] Joã|o M. S. Silva1, Jorge Fernández Villena, Paulo Flores, and L. Miguel.
Outstanding issues in model order reduction.
In *Scientific Computing in Electrical Engineering*, pages 139–152, 2006.

[18] Swaminathan Subramanian, Dhananjai Madhava Rao, and Philip A Wilsey.
Study of a multilevel approach to partitioning for parallel logic simulation.
In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pages 833–838. IEEE, 2000.

[19] V.Oppenhiem, S.Willsky, and S.Hamid.

Signals and Systems.
Pearson-2nd Edition, 1996.

[20] JIA Zhongxiao.
Composite orthogonal projection methods for large matrix eigenproblems.
Science in China Series A: Mathematics, 42(6):577–585, 1999.