M. Phil. Thesis

AN AUTOMATED SYSTEM TO ANALYZE SYSTEM DYNAMICS MODELS

By

Ahmed AbdelTawab AbdelGawad

Supervisors

Prof. Pål I. Davidsen
Dr. Mohamed M. Saleh

Department of Information Science
University of Bergen

2004

ABSTRACT

For a very long time, software oriented to analyze system dynamics models using eigenvalue analysis technique was not more than a dream. System Dynamics has been hampered by the lack of such software used to analyze the relationship between the structure and behavior in complex, dynamic models automatically.

In this thesis, An A to Z mathematical background has been developed, based on Control Theory literature as well as the previous work in the filed of applying eigenvalue analysis to the system dynamics models[*]*, this is in addition to the development of a Matlab code to automate the examination process of the structural origin of different modes of behavior exhibited by a system dynamics model using mathematical method crystallized in the mathematical background. This method allows for an investigation of how model behavior is created from the underlying model structure and how this behavior feeds back to change the relative significance of the model behavior. They also allow us to identify the dynamics of relative significance of the various parameters that governs the gains of the links and loops of the model.*

By automating this method into Matlab code, System Dynamists have the luxury of behavior to structure identification in a fast and an accurate way that can be further implemented as a part of the simulation package to make the analysis an intermediate process through the modeling process.

[*] Nathan B. Forrester, Christian C. Kampmann, Mohamed M. Saleh and Pål I. Davidsen.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Pål I. Davidsen and Dr. Mohamed M. Saleh for their assistance in the preparation of this research.

In addition, special thanks to Dr. Nabil Saeed who gave me the chance to study for this degree in the first place.

Thanks to my close friend Bahaa El-Din Ali who the least I can say about him is "a friend in need is a friend indeed".

Thanks to my sister Assmaa for giving me the idea of using the Data Flow Diagram to enhance the implementation chapter, and helping me in drawing the diagrams.

Thanks to my friend Essam AbdelMottaleb for his great effort in revising the mathematical background and the application chapters.

Thanks to the members and management of the Information and Decision Support Center for their help through the whole research and thesis articulation, especially Reem AbdelHaliem and Sherine Haikal for their sincere effort in the revision of the thesis.

Thanks to my parents and sisters who have been encouraging me to finish this work all the time, without their encouragement this work was not to be done.

To My Parents

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

Introduction and Literature Review

1.1 Introduction

The real benefit of a system dynamics model cannot be crystallized until it is possible to determine the causes of the behavior observed in this model, So that an extension to System Dynamics emerges from the Control Theory and Mathematics to deeply understand the model behavior as well as discovering the dominant modes in that behavior and the dominant structures that cause them.

Before going any further, a set of definition is essential. These definitions aim at answering three basic questions:

- What is a model?
- What is system dynamics?
- What is the eigenvalue analysis?

1.1.1 What is a model?

An interesting and very descriptive definition by Geoffrey Gordon in his book "System Simulation", the second edition 1989:

> *"We define a model as the body of information about a system gathered for the purpose of studying the system."*
>
> *– Geoffrey Gordon (1989)*

Another dictionary definition of a model that describes the model more deeply from the structure point of view:

> *"A simplified representation of a system or phenomenon, as in the sciences or economics, with any hypotheses required to describe the system or explain the phenomenon, often mathematically."*
>
> *– Webster's Electronic Dictionary and Thesaurus*

1.1.2 What is system dynamics?

In the 1950s, Jay W. Forrester developed the System dynamics which is a branch of modeling deals with simulation models.

> *"System dynamics is an approach to the study of complexity. Originally developed at the Massachusetts Institute of Technology by Jay Forrester, system dynamics is a unique method devised to help managers and public policymakers design and implement high leverage policies for sustainable success."*
>
> *– The back cover of Business Dynamics by John D. Sterman (1999)*

And as Forrester tells in his paper about the new product diffusion in an open market; trying to explain the true purpose of system dynamics:

> *"One can identify a system only in terms of an objective. Here the objective is to identify and to explain one of the systems which can cause stagnation of sales growth even in the presence of an unlimited market. In particular, we deal here with that system which causes sales stagnation, or even sales decline, to arise out of an overly cautious capital investment policy. In this system inadequate capacity limits the growth in product sales."*

– Jay W. Forrester (1975)

1.1.3 Eigenvalue Analysis

> *"Eigenvalue analysis of dominant feedback loops promises to be a powerful new tool for identifying the structural origins of behavior in system dynamics models."*
>
> *– Nathan B. Forrester (1983)*

Eigenvalue analysis is a mathematical method developed to identify the leverage points in a model, without having to do tedious and erroneous simulation experiments.

1.2 Literature review

The contribution in the field of system dynamics models analysis although various, it is considerably modest; which is considered a result of the discontinuous nature of the research in this field.

Research in system dynamics models analysis started in 1982, by the introduction of Eigenvalue analysis approach by Nathan B. Forrester in his Ph.D. thesis, introduced to the M.I.T.; which was a prosperous start. He continued his work by a paper introduced in 1983 to the 1983 International System Dynamics Conference in Massachusetts, Nathan Forrester was the first to look back in the origin of the system dynamics science, the control theory, and tries to adapt yet other mathematical methods to help system dynamists, to analyze their models and interesting ways to find out new stabilizing policies.

Leaving a very wide step of thirteen years, and in the 1996 International System Dynamics Conference, Christian C. Kampmann introduced his interesting paper to compete what Nathan Forrester started and towards the completeness of the whole work, he introduced other interesting mathematical approaches to enrich the research of system dynamics models analysis, like the Graph or Network Theory to express system dynamics model in a matrix form that can be searched for paths and loops, and it was considered a great step forward.

Another wide step of nearly four years, another stream started, and in 2000 and 2001 Mohamed M. Saleh and Pål I. Davidsen introduced a more complete piece of work to the system dynamics conference 2000, 2001. They paved the road that starts at the system dynamics models and ends at the models Eigenvalue full analysis. This was concluded nearly totally with Saleh's Ph.D. thesis under the supervision of Davidsen introduced to the University of Bergen.

And one step is still needed, this step has been demanded, starting from Nathan Forrester and ending by Saleh. There should be an automated way to complete the task of Eigenvalue analysis, i.e. a computer package that takes the system dynamics model, and make the whole analysis work and introduce the needed results to the modeler, who by all means in dispense with the tedious mathematical processes.

1.2.1 The Work of Nathan B. Forrester

In his Ph.D. thesis (MIT – 1983) Nathan B. Forrester introduced the eigenvalue analysis and frequency response techniques applied to his US economy system dynamics model. He aimed at producing a proper stabilizing policy for his model.

Then in his paper submitted to the international System Dynamics Conference (1984), he continued his work, but this time he was very specific and concentrated on the eigenvalue analysis techniques as a method to analyze loop dominance, and compared the eigenvalue analysis with other two traditional approaches used to do the same task. Moreover he introduced to model linear approximation using Taylor series to convert a nonlinear model into a linear one to give the possibility to apply eigenvalue analysis on nonlinear models.

He related each eigenvalue identified in the model with one of the produced modes of behavior that constitutes the total behavior of that model.

In another words he related the model structure with value of the eigenvalue using the eigenvalue elasticity concept.

1.2.2 The Work of Christian C. Kampmann

In his valuable paper submitted to the International System Dynamics Conference (1996), he introduced to the expression of the system dynamics model as digraph. Moreover he introduced the linearly independent loop set, and he could solve a system of equations to find their eigenvalue elasticity values.

1.2.3 The Work of Mohamed M. Saleh and Pål I. Davidsen

In their paper submitted to the International System Dynamics Conference (2000), and in the thesis of Mohamed M. Saleh, which he submitted to the University of Bergen (2003) under the supervision of Pål I. Davidsen, a full mathematical framework was introduced to complete the story of the eigenvalue analysis.

1.3 Purpose of the Research

The purpose of this research could be divided into three different pieces:

1. Putting a standard (symbols and names) for the mathematical presentation needed for the eigenvalue analysis that complies with those of the control theory
2. Developing a full user-friendly software package, that implements the eigenvalue analysis technique
3. Analyzing the market growth model, and identifying the leverage points in that model

Chapter 2

The Analysis Package: Mathematical Background

2.1 Introduction

This chapter investigates in details; the mathematical foundation of the eigenvalue analysis, in terms of its steps applied to system dynamics models.

Section 2.2 discusses in brief the system dynamics model from its equations perspective, besides introducing a definition for the inputs and the outputs of the model and explaining how to identify inputs as well as choosing outputs.

Section 2.3 defines the state variables and identifies them in the system dynamics model.

Section 2.4 goes back again to the model, this time from the linearity and nonlinearity perspective; and introduces the definitions of linear and nonlinear model, in addition to introducing a mathematical way to convert a nonlinear model into a linear one.

Section 2.5 shows how to put the state equations into the state space form.

Section 2.6 identifies the characteristic equation and then eigenvalues of the model from the state space form. The eigenvalues of the model are discussed in details starting by their nature as complex numbers and ending by their corresponding different modes of behavior identified in model behavior.

Section 2.7 shows how to identify the dominant eigenvalue on the behavior of some state in the model.

Section 2.8 concerns on computing the dominant eigenvalue elasticity values for every link, constant and linearly independent loop in the model.

Section 2.8.6 discusses loops and linearly independent loops of the model and their dominant eigenvalue elasticity values.

2.2 System Dynamics Models

"A little learning (or knowledge) is a dangerous thing."

–Pope, Alexander (1688 - 1744).

Identifying a system dynamics model from the stock and flow perspective is not enough. The core of the model lies in the equations that contain dynamics. Its limbs are its inputs and outputs where external world can deal with it and where it can respond.

Before going into the system dynamics model eigenvalue analysis steps, two important issues about system dynamics model need to be discussed:

1. Equations.
2. Inputs and outputs.

2.2.1 System Dynamics Model's Equations

By setting the stock and flow diagram aside, and giving a look at the equation view in any system dynamics simulation software package, we would identify that; the model consists of a set of mathematical equations that build up the dynamic system. Those equations are of two types:

1. A set of first order differential equations, defining the dynamics in the dynamic system (the levels' equations).
2. A set of algebraic equations, defining the static relations in the dynamic system (the auxiliaries' equations).

Collectively we can say that the system dynamics model is a set of first order differential equations with embedded static relations (the algebraic equations).

Figure 1 shows the stock and flow of the yeast cells model, this model explains the life cycle of the yeast cells; every cell produces alcohol as long as being alive. It also, reproduces through cell division process. The problem exists in that as the alcohol concentration increases the cells deaths, and their population decreases till totally vanishes. Table 1 shows the equations listing of the yeast cells model, and from it we would notice that the first two equations are integral equations (levels), while the rest are algebraic equations (auxiliaries) followed by the values of the inputs (constants).

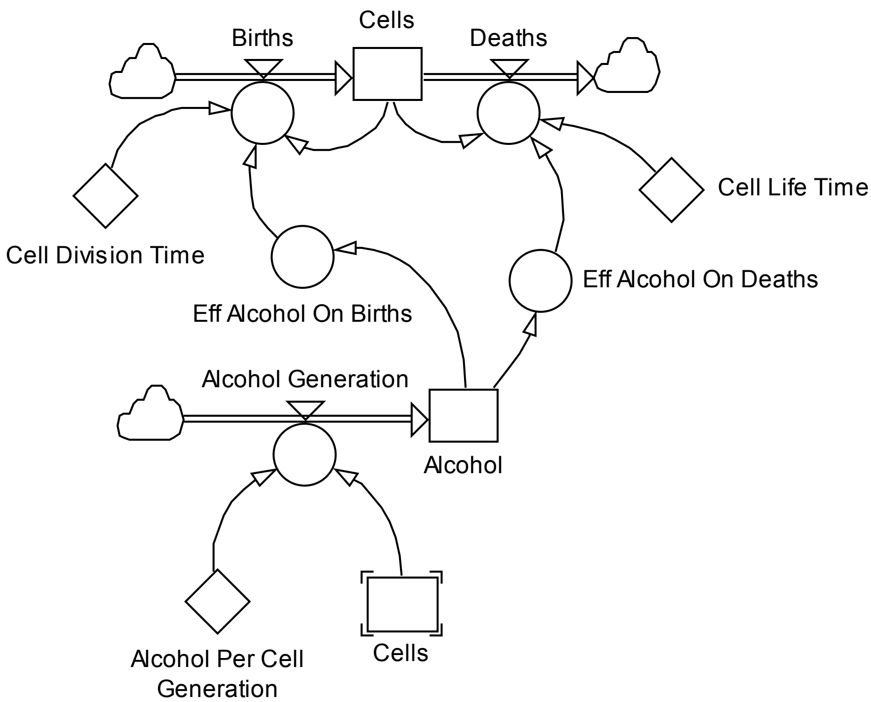


Figure 1. The Yeast Cells Model

1	☐ Alcohol ◇ 0 ⇨ +dt*Alcohol Generation 🌡 Milliliters
2	☐ Cells ◇ 1 ⇦ -dt*Deaths ⇨ +dt*Births 🌡 Cells
3	⇨ Alcohol Generation ○ Cells*Alcohol Per Cell Generation 🌡 Milliliters/Minutes
4	⇨ Births ○ Cells*Eff Alcohol On Births/Cell Division Time 🌡 Cells/Minutes
5	⇨ Deaths ○ Cells*Eff Alcohol On Deaths/Cell Life Time 🌡 Cells/Minutes
6	○ Eff Alcohol On Births ○ (-0.1*Alcohol)+1.1 🌡 Dimensionless
7	○ Eff Alcohol On Deaths ○ exp(Alcohol-11) 🌡 Dimensionless
8	◇ Alcohol Per Cell Generation ◇ 0.01 🌡 Milliliters/Cells/Minutes
9	◇ Cell Division Time ◇ 15 🌡 Minutes
10	◇ Cell Life Time ◇ 30 🌡 Minutes

Table 1: The Yeast Cells Model's Equations

The first equation in Table 1:

$$Alcohol \ = \ INTEGRAL(Alcohol\ Generation,\ 0)$$

This can be rewritten in mathematical form:

$$Alcohol\Big|_0^{Alcohol} = \int_{t_0}^{t} Alcohol\ Generation \cdot dt$$

By differentiating both sides:

$$\frac{d}{dt}\left(Alcohol\right) = Alcohol\ \ Generation \qquad (1)$$

The second equation in Table 1:

$$Cells \ = \ INTEGRAL(Births - Deaths,\ 1)$$

This can be rewritten in a mathematical form:

$$Cells\Big|_1^{Cells} = \int_{t_0}^{t} \left(Births - Deaths\right) \cdot dt$$

By differentiating both sides:

$$\frac{d}{dt}\left(Cells\right) = Births - Deaths \qquad (2)$$

The terms $\frac{d}{dt}\left(Alcohol\right)$ and $\frac{d}{dt}\left(Cells\right)$ are the net flow rates to $Alcohol$ and $Cells$ level variables respectively, and both of their equations are differential and of the first order.

2.2.2 System Dynamics Model's Inputs and Outputs

In the control theory, models are classified as Single Input Single Output **SISO** or Multi Input Multi Output **MIMO** (Kuo, B. C., 1995). System dynamics models can be classified the same way, after identifying their inputs and choosing their outputs.

2.2.2.1Identifying the Inputs

According to the concept of inputs in the control theory, inputs of the model are these influences (variables) that act on the model from outside and are not affected by what happens inside it (Kheir, Naim A., 1996). Exactly this is the definition of the constants in a system dynamics model –of course, except the initial values of the levels–, so that the inputs of the system dynamics model can be any collection of its constants. This is decided by the modeler –reflection of the system in his/her mind– because modelers may like to have constants that would never change inside the model time span, so that they can not be accepted as inputs.

For yeast cells model, the inputs might be any collection of the following model constants: $Alcohol\ Per\ Cell\ Generation$, $Cell\ Division\ Time$ or $Cell\ Life\ Time$. Then the chosen set would be placed in a vector that is called the input vector and has the symbol $\mathbf{u}$ [†], for example, if we take all the previous list of constants to be inputs, the input vector would be:

$$\mathbf{u} = \begin{bmatrix} Alcohol\ Per\ Cell\ Generation \\ Cell\ Division\ Time \\ Cell\ Life\ Time \end{bmatrix} \tag{3}$$

[†] Mathematical symbol for a vector will always be a lower case bold non-italic letter like $\mathbf{x}$, mathematical symbol for a matrix will always be an upper case bold non-italic letter like $\mathbf{X}$ and mathematical symbol for an element of a vector or a matrix will always be a lower case non-bold italic letter with index as subscripts like x_{ij}

2.2.2.2Choosing the Outputs

Again according to the concept of inputs in control theory, output variables are observable quantities and are measurable (Kheir, Naim A., 1996). In system dynamics model this definition is still valid, and outputs are always the choice of the user or the modeler. They can be any collection of the variables of the model.

For Yeast Cell model, the outputs might be any collection of the following list of variables: $Alcohol$, $Cells$, $Alcohol\ Generation$, $Eff\ \ Alcohol\ \ On\ \ Births$, $Eff\ \ Alcohol\ \ On\ \ Deaths$, $Births$ and $Deaths$.

The chosen set would be placed in a vector that is called the output vector and has the symbol $\mathbf{y}$, for example, if we take $Alcohol$, $Cells$ and $Alcohol\ Generation$ to be the outputs, the output vector will be:

$$\mathbf{y} = \begin{bmatrix} Alcohol \\ Cells \\ Alcohol\ Generation \end{bmatrix} \tag{4}$$

2.3 State Variables and State Equations and Other Equations

Before going any further, the definition of State variables of dynamic model should be very clear.

> *"The state of a system refers to the past, present, and future conditions of the system. From a mathematical sense, it is convenient to define a set of state variables and state equations to model dynamic systems."*
>
> *–Kuo, B. C. (1995).*

> *"The state variables must satisfy the following conditions:*
>
> *At any initial time $t = t_0$, the state variables $x_1(t_0)$, $x_2(t_0)$, … , $x_n(t_0)$ define the initial state of the system.*
>
> *Once the inputs of the system for $t > t_0$ and the initial state defined above are specified, the state variables should completely define the future behavior of the system."*
>
> *–Kuo, B. C. (1995).*

> *"The state of a dynamic system is the smallest set of variables (called state variables) such that the knowledge of these variables at $t = t_0$, together with knowledge of the input for $t \geq t_0$, completely determines the behavior of the system for any time $t \geq t_0$."*
>
> *–Ogata, K. (1997).*

This concludes that the state variables are the smallest set variables that define the state of the dynamic system, at the initial time. Also, the state of the system in the future depends on their present values.

These definition and conditions of State variables are conformable with that of the levels variables of the system dynamics model.

Back to the yeast cells model, the state variables are: $Alcohol$ and $Cells$, and their differential equations are the state equations of the model equations (1) and (2).

As we did with inputs and outputs, we would place the states in a vector and call it the state vector and give it the symbol $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} Alcohol \\ Cells \end{bmatrix} \tag{5}$$

In addition the other variables of the model –the auxiliary variables– would be placed in a vector form, this vector would have the symbol $\mathbf{z}$:

$$\mathbf{z} = \begin{bmatrix} Alcohol\ Generation \\ Births \\ Deaths \\ Eff\ \ Alcohol\ On\ Births \\ Eff\ \ Alcohol\ On\ Deaths \end{bmatrix} \tag{6}$$

The time derivative of state vector –the net rates vector– would be:

$$\dot{\mathbf{x}} = \frac{d}{dt}\mathbf{x} = \begin{bmatrix} \frac{d}{dt}(Alcohol) \\ \frac{d}{dt}(Cells) \end{bmatrix} \tag{7}$$

It is noticeable that the net rates do not have distinct names for themselves; they have their names from the states they are connected to. Also, the rates (not the net rates) are considered to be of the auxiliary variables, so that and from equations (1) and (2), we will find that the net rates are some auxiliary variables added or subtracted to or from each others, i.e. net rates are polynomials of the first degree of auxiliary variables.

2.4 Linear vs. Nonlinear Models

From the design and the analysis point of view, models are **Linear** or **Nonlinear**. And although linear systems don't exist in practice, nearly all analysis methods in control theory are based on the assumption that systems are linear.

2.4.1 Linear Models

A dynamic system is called linear when the principle of superposition holds. Meaning that; the model response to the change in several parameters can be calculated by changing one parameter at a time and adding the results. Also if cause and effect are proportional, this means that the model is linear (Ogata, K., 1997).

The model is said to be linear, if the following equation holds for all equations of its auxiliary variables as well as net rates,

$$z_h = a_1 x_1 + \cdots + a_{N_x} x_{N_x} + b_1 z_1 + \cdots + b_{N_z} z_{N_z} + c_1 u_1 + \cdots + c_{N_u} u_{N_u} \qquad (8)$$

Where: $x_i : i \in \mathbb{Z}^+ \leq N_x$ ‡, $z_j : j \in \mathbb{Z}^+ \leq N_z$ and $u_k : k \in \mathbb{Z}^+ \leq N_u$ are the level, auxiliary and input variables respectively, $a_i : i \in \mathbb{Z}^+ \leq N_x$, $b_j : j \in \mathbb{Z}^+ \leq N_z$ and $c_k : k \in \mathbb{Z}^+ \leq N_u$ are constants and N_x, N_z and N_u are the number of level, auxiliary and input variables respectively.

By expressing every variables in the model as a deviation from chosen specific initial operating point that is selected on its behavior, i.e. by replacing x_1, … , x_{N_x}, z_1, … , z_{N_z}, u_1, … and u_{N_u} by: $\tilde{x}_1 + \delta x_1$, … , $\tilde{x}_{N_x} + \delta x_{N_x}$, $\tilde{z}_1 + \delta z_1$, … , $\tilde{z}_{N_z} + \delta z_{N_z}$, $\tilde{u}_1 + \delta u_1$, … and $\tilde{u}_{N_u} + \delta u_{N_u}$ respectively.

‡ The Set of Positive Integers 1, 2, 3, ... , denoted $\mathbb{Z}^+$ (Weisstein, E. W. (1999) Concise Encyclopedia of Mathematics CD-ROM).

While those δ terms are very small values, and the values having tilde over them are the chosen specific initial operating point values, then also z_h would be expressed as $\tilde{z}_h + \delta z_h$:

$$\begin{aligned}\therefore \tilde{z}_h + \delta z_h = a_1(\tilde{x}_1 + \delta x_1) + \ldots + a_{N_x}(\tilde{x}_{N_x} + \delta x_{N_x}) \\ + b_1(\tilde{z}_1 + \delta z_1) + \ldots + b_{N_z}(\tilde{z}_{N_z} + \delta z_{N_z}) \\ + c_1(\tilde{u}_1 + \delta u_1) + \ldots + c_{N_u}(\tilde{u}_{N_u} + \delta u_{N_u})\end{aligned}$$

$$\begin{aligned}\therefore \tilde{z}_h + \delta z_h = (a_1\tilde{x}_1 + \ldots + a_{N_x}\tilde{x}_{N_x} + b_1\tilde{z}_1 + \ldots + b_{N_z}\tilde{z}_{N_z} + c_1\tilde{u}_1 + \ldots + c_{N_u}\tilde{u}_{N_u}) \\ + (a_1\delta x_1 + \ldots + a_{N_x}\delta x_{N_x} + b_1\delta z_1 + \ldots + b_{N_z}\delta z_{N_z} + c_1\delta u_1 + \ldots + c_{N_u}\delta u_{N_u})\end{aligned}$$

Taking into consideration that the originally chosen initial operating point should be selected from the behavior of the $\tilde{z}_h$, i.e. it satisfies the original equation of $\tilde{z}_h$. In terms of mathematical equations:

$$\tilde{z}_h = a_1\tilde{x}_1 + \cdots + a_n\tilde{x}_n + b_1\tilde{z}_1 + \cdots + b_m\tilde{z}_m + c_1\tilde{u}_1 + \cdots + c_l\tilde{u}_l$$

$$\therefore \delta z_h = a_1\delta x_1 + \cdots + a_n\delta x_n + b_1\delta z_1 + \cdots + b_m\delta z_m + c_1\delta u_1 + \cdots + c_l\delta u_l \qquad (9)$$

2.4.2 Nonlinear Models

Generally all systems are nonlinear. So that, most practical models have nonlinear relationships among their variables, which implies that those models are nonlinear and that equation (8) does not hold for all their variables, but they take the following form:

$$z_h = f(x_1, \ldots, x_{N_x}, z_1, \ldots, z_{N_z}, u_1, \ldots, u_{N_u}) \qquad (10)$$

Where: $f(.)$ is a nonlinear function.

But as stated previously, the analysis process requires the model equations to be linear, so that modifications to the equations of the nonlinear model is needed to change their nonlinear nature into linear, but under an important condition that is not to change the model behavior.

2.4.3 Model Linearization

The model linearization processes is the process of changing the model from nonlinear into linear, i.e. to put variables having the form of equation (10) in the form of equation (8) or (9), without changing the behavior within a limitation boundaries, and this is possible using **Taylor Series**.

If we have a model that has an equation like equation (10), which has a nonlinear part, by expressing all variables of the model as a deviation from chosen specific normal operating point that is selected from the behavior of z_h, i.e. by replacing $x_1, \ldots, x_{N_x}, z_1, \ldots, z_{N_z}, u_1, \ldots$ and u_{N_u} by: $\tilde{x}_1 + \delta x_1, \ldots,$ $\tilde{x}_{N_x} + \delta x_{N_x}$, $\tilde{z}_1 + \delta z_1$, $\ldots$, $\tilde{z}_{N_z} + \delta z_{N_z}$, $\tilde{u}_1 + \delta u_1$, $\ldots$ and $\tilde{u}_{N_u} + \delta u_{N_u}$ respectively.

While those δ terms are very small values, and the values having tilde over them are the chosen specific initial operating point values, then z_h would also be expressed as $\tilde{z}_h + \delta z_h$, where originally the chosen initial operating point should be selected from the behavior of the $\tilde{z}_h$, i.e. it satisfies the original equation of $\tilde{z}_h$. In terms of mathematical equations:

$$\tilde{z}_h = f\,(\tilde{x}_1,\ldots,\tilde{x}_{N_x},\tilde{z}_1,\ldots,\tilde{z}_{N_z},\tilde{u}_1,\ldots,\tilde{u}_{N_u})$$

$$\begin{aligned}\therefore \tilde{z}_h + \delta z_h = f\,(&\tilde{x}_1 + \delta x_1,\ldots,\tilde{x}_{N_x} + \delta x_{N_x},\\ &\tilde{z}_1 + \delta z_1,\ldots,\tilde{z}_{N_z} + \delta z_{N_z},\\ &\tilde{u}_1 + \delta u_1,\ldots,\tilde{u}_{N_u} + \delta u_{N_u})\end{aligned} \qquad (11)$$

Equation (11) could be expanded using Taylor Series:

$$\begin{aligned}\tilde{z}_h + \delta z_h = {} & f(\tilde{x}_1,\ldots,\tilde{x}_{N_x},\tilde{z}_1,\ldots,\tilde{z}_{N_z},\tilde{u}_1,\ldots,\tilde{u}_{N_u}) \\ & + \left.\frac{\partial f}{\partial x_1}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta x_1 + \ldots + \left.\frac{\partial f}{\partial x_{N_x}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta x_{N_x} \\ & + \left.\frac{\partial f}{\partial z_1}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta z_1 + \ldots + \left.\frac{\partial f}{\partial z_{N_z}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta z_{N_z} \\ & + \left.\frac{\partial f}{\partial u_1}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta u_1 + \ldots + \left.\frac{\partial f}{\partial u_{N_u}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta u_{N_u} \\ & + H.O.T.\end{aligned}$$

Where: $\tilde{\mathbf{x}} = \begin{bmatrix}\tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_{N_x}\end{bmatrix}$, $\tilde{\mathbf{z}} = \begin{bmatrix}\tilde{z}_1 \\ \tilde{z}_2 \\ \vdots \\ \tilde{z}_{N_z}\end{bmatrix}$, $\tilde{\mathbf{u}} = \begin{bmatrix}\tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_{N_u}\end{bmatrix}$ and $H.O.T.$ is the total amount of the higher order terms, taking into consideration that those δ terms were assumed to be very small values, the higher order terms $H.O.T.$ other than the first order terms would have very small values, and can be ignored compared to those of the first order terms. Also $\tilde{z}_h = f(\tilde{x}_1,\ldots,\tilde{x}_{N_x},\tilde{z}_1,\ldots,\tilde{z}_{N_z},\tilde{u}_1,\ldots,\tilde{u}_{N_u})$; the last equation could be reduced to:

$$\delta z_h = \sum_{i=1}^{N_x} \left.\frac{\partial f}{\partial x_i}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta x_i + \sum_{j=1}^{N_z} \left.\frac{\partial f}{\partial z_j}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta z_j + \sum_{k=1}^{N_u} \left.\frac{\partial f}{\partial u_k}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta u_k \tag{12}$$

Where: $\left.\frac{\partial f}{\partial x_i}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} : i \in \mathbb{Z}^+ \le N_x$, $\left.\frac{\partial f}{\partial z_j}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} : j \in \mathbb{Z}^+ \le N_z$ and $\left.\frac{\partial f}{\partial u_k}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} : k \in \mathbb{Z}^+ \le N_u$ are all constants, and could be replaced by $a_i : i \in \mathbb{Z}^+ \le N_x$, $b_j : j \in \mathbb{Z}^+ \le N_z$ and $c_k : k \in \mathbb{Z}^+ \le N_u$ respectively.

$$\delta z_h = a_1 \delta x_1 + \cdots + a_{N_x} \delta x_{N_x} + b_1 \delta z_1 + \cdots + b_{N_z} \delta z_{N_z} + c_1 \delta u_1 + \cdots + c_{N_u} \delta u_{N_u} \tag{13}$$

By comparing equations (10) and (13) –taking into consideration that the δ terms represents a small deviation (change) in the original term; i.e. still expressing the original term if the initial value of the original term is exactly known–, it should be noticeable that the nonlinear relation could be replaced with a linear one.

2.5 State Space Form

At this point, we want to put those state equations of the linear or the linearized model in a general matrix form that is suitable for the analysis process, this form is called the state space from.

By applying equation (12) on the h[th] element in the net rates vector –as previously clarified, the net rate is a polynomial of the first degree of auxiliary variables–:

$$\delta \dot{x}_h = \sum_{i=1}^{N_x} \left. \frac{\partial f}{\partial x_i} \right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta x_i + \sum_{j=1}^{N_z} \left. \frac{\partial f}{\partial z_j} \right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta z_j + \sum_{k=1}^{N_u} \left. \frac{\partial f}{\partial u_k} \right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta u_k \tag{14}$$

Taking into consideration that $h \in \mathbb{Z}^+ \leq N_x$, it would be obviously noticed that the three summations represents matrix multiplication results; so that, equation (14) can be rewritten to be:

$$\delta \dot{\mathbf{x}} = \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} \Big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta \mathbf{x} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} \Big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta \mathbf{z} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}} \Big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta \mathbf{u} \tag{15}$$

Where: $\delta\dot{\mathbf{x}} = \begin{bmatrix} \delta\dot{x}_1 \\ \delta\dot{x}_2 \\ \vdots \\ \delta\dot{x}_{N_x} \end{bmatrix}$, $\delta\mathbf{x} = \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \\ \delta x_{N_x} \end{bmatrix}$, $\delta\mathbf{z} = \begin{bmatrix} \delta z_1 \\ \delta z_2 \\ \vdots \\ \delta z_{N_z} \end{bmatrix}$ and $\delta\mathbf{u} = \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \vdots \\ \delta u_{N_u} \end{bmatrix}$ are the deviations in the net rates, level variables, auxiliary variables and the input variables vectors respectively.

Also, $\mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\dot{\mathbf{x}}}{\partial\mathbf{x}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$, $\mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\dot{\mathbf{x}}}{\partial\mathbf{z}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ and $\mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\dot{\mathbf{x}}}{\partial\mathbf{u}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$, and those $\mathbf{J}$*'s* are called the **Jacobian**[§].

By applying equation (12) on the g[th] element in the auxiliary variables vector:

$$\delta z_g = \sum_{i=1}^{N_x} \left.\frac{\partial f}{\partial x_i}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta x_i + \sum_{j=1}^{N_z} \left.\frac{\partial f}{\partial z_j}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta z_j + \sum_{k=1}^{N_u} \left.\frac{\partial f}{\partial u_k}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta u_k \tag{16}$$

And again, $g \in \mathbb{Z}^+ \le N_x$. So that, the three summations represents matrix multiplication results; as a result, equation (16) can be rewritten to be:

$$\delta\mathbf{z} = \mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{z} + \mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u} \tag{17}$$

Where: $\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\mathbf{z}}{\partial\mathbf{x}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$, $\mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\mathbf{z}}{\partial\mathbf{z}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ and $\mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\frac{\partial\mathbf{z}}{\partial\mathbf{u}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$.

Equations (15) and (17) could be merged in the following form:

$$\begin{bmatrix} \delta\dot{\mathbf{x}} \\ \hdashline \delta\mathbf{z} \end{bmatrix} = \left.\begin{bmatrix} \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} & \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} \\ \mathbf{J}_{\mathbf{z},\mathbf{x}} & \mathbf{J}_{\mathbf{z},\mathbf{z}} \end{bmatrix}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \begin{bmatrix} \delta\mathbf{x} \\ \hdashline \delta\mathbf{z} \end{bmatrix} + \left.\begin{bmatrix} \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}} \\ \hdashline \mathbf{J}_{\mathbf{z},\mathbf{u}} \end{bmatrix}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} \delta\mathbf{u} \tag{18}$$

---

§ Named after the German mathematician CARL GUSTAV JACOB JACOBI (Kreyszig, E., 1993).

$$\mathbf{J} = \frac{\partial([x,y])}{\partial([u,v])} = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{bmatrix}$$

The matrix $\left[\begin{array}{c|c} \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} & \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} \\ \hline \mathbf{J}_{\mathbf{z},\mathbf{x}} & \mathbf{J}_{\mathbf{z},\mathbf{z}} \end{array}\right]\Bigg|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ relates all the variables of the model to each other using their gain values, so that it is called the **System Jacobian** (Kampmann, C. E., 1996), or it could be called the **Full Gain Matrix** after the full version of the model –in contrast with the **Compact Gain Matrix** of the compact version of the model (Saleh, M.; Davidsen, P. I., 2000) – and because it contains the gains of all model links.

Also, from equation (17):

$$\delta\mathbf{z} = \mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{z} + \mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}$$

$$\therefore \delta\mathbf{z} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{z} = \mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + \mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}$$

$$\therefore (\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})\delta\mathbf{z} = \mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + \mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}$$

$$\therefore \delta\mathbf{z} = (\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + (\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}$$

By substituting in equation (15):

$$\begin{aligned}\therefore \delta\dot{\mathbf{x}} &= \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} \\ &+ \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\left((\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + (\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}\right) \\ &+ \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}\end{aligned}$$

$$\begin{aligned}\therefore \delta\dot{\mathbf{x}} &= \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,(\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{x} \\ &+ \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,(\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,\delta\mathbf{u}\end{aligned}$$

$$\begin{aligned}\therefore \delta\dot{\mathbf{x}} &= \left(\mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,(\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\right)\delta\mathbf{x} \\ &+ \left(\mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,(\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\right)\delta\mathbf{u}\end{aligned}$$

By putting $\mathbf{A} = \mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}\,(\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1}\,\mathbf{J}_{\mathbf{z},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$

and $\mathbf{B} = \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} (\mathbf{I} - \mathbf{J}_{\mathbf{z},\mathbf{z}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}})^{-1} \mathbf{J}_{\mathbf{z},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} + \mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$:

$$\delta\dot{\mathbf{x}} = \mathbf{A}\delta\mathbf{x} + \mathbf{B}\delta\mathbf{u} \tag{19}$$

In control theory context the matrix $\mathbf{A}$ is called the **System Matrix**, while in system dynamics context it would be the **Compact Gain Matrix** as stated previously. The matrix $\mathbf{B}$ is called the **Input Matrix** or **Control Matrix**.

Using the same method used with the last equation:

$$\delta\mathbf{y} = \mathbf{C}\delta\mathbf{x} + \mathbf{D}\delta\mathbf{u} \tag{20}$$

Where: $\delta\mathbf{y} = \begin{bmatrix} \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_{N_y} \end{bmatrix}$ is the output vector. $\mathbf{C}$ and $\mathbf{D}$ are the **Output Matrix** and **Feedforward Matrix** respectively –control theory context–.

Also, $\mathbf{C} = \mathbf{J}_{\mathbf{y},\mathbf{x}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\dfrac{\partial\mathbf{y}}{\partial\mathbf{x}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ and $\mathbf{D} = \mathbf{J}_{\mathbf{y},\mathbf{u}}\big|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}} = \left.\dfrac{\partial\mathbf{y}}{\partial\mathbf{u}}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$.

Back to the yeast cells example:

$\mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} = \left.\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$, and this is normal, because we deal with the net rates as a polynomial of auxiliary variables, i.e. there is no direct connections from $\mathbf{x}$ elements to $\dot{\mathbf{x}}$ elements, but the connections are through $\mathbf{z}$ elements –as discussed before–.

$$\mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} = \left.\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

$$\mathbf{J}_{\mathbf{z},\mathbf{x}} = \begin{bmatrix} \text{-0.1} & 0 \\ \exp\left(Alcohol\ \text{-}11\right) & 0 \\ 0 & 1/100 \\ 0 & 1/15*Eff\ \ Alcohol\ \ On\ \ Births \\ 0 & 1/15*Eff\ \ Alcohol\ \ On\ \ Births \end{bmatrix}_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

$$\mathbf{J}_{\mathbf{z},\mathbf{z}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1/15*Cells & 0 & 0 & 0 & 0 \\ 0 & 1/30*Cells & 0 & 0 & 0 \end{bmatrix}_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

$\mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ and this is normal too, for the same reason discussed previously for the $\mathbf{J}_{\dot{\mathbf{x}},\mathbf{u}}$.

The full gain matrix can be identified easily by joining the pervious $\mathbf{J}$'s.

The system matrix or compact gain matrix easily computed:

$$\mathbf{A} = \begin{bmatrix} 0 & 1/100 \\ \left(\text{-}1/150*Cells\right. & \left(1/15*Eff\ \ Alcohol\ \ On\ \ Births\right. \\ \left.\text{-}1/30*Cells*\exp\left(Alcohol\ \text{-}11\right)\right) & \left.\text{-}1/30*Eff\ \ Alcohol\ \ On\ \ Deaths\right) \end{bmatrix}_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

And, the input matrix according to the previously chosen input vector:

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & Cells \\ \left(\text{-}1/225*Cells\right. & \left(1/900*Cells\right. & \\ \left.*Eff\ \ Alcohol\ \ On\ \ Births\right) & \left.*Eff\ \ Alcohol\ \ On\ \ Deaths\right) & 0 \end{bmatrix}_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

All those matrices would be evaluated at any selected operating point at $\tilde{\mathbf{x}}$, $\tilde{\mathbf{z}}$ and $\tilde{\mathbf{u}}$ to complete the eigenvalue analysis process.

Back to equation (19), and by dividing both sides by δt which represents a very small time change. And by taking limits to both sides while δt approaches 0:

$$\lim_{\delta t \to 0} \frac{\delta \dot{\mathbf{x}}}{\delta t} = \mathbf{A} \lim_{\delta t \to 0} \frac{\delta \mathbf{x}}{\delta t} + \mathbf{B} \lim_{\delta t \to 0} \frac{\delta \mathbf{u}}{\delta t}$$

From the definition of the differentiation:

$$\therefore \ddot{\mathbf{x}} = \mathbf{A}\dot{\mathbf{x}} + \mathbf{B}\dot{\mathbf{u}} \qquad (21)$$

At this point we are assuming that $\mathbf{A}$ and $\mathbf{B}$ are constants and don't include any functions of time.

Under normal model simulation conditions, the $\mathbf{u}$ vector is a constant vector all the time, which implies that $\dot{\mathbf{u}}$ equals $\mathbf{0}$.

$$\therefore \ddot{\mathbf{x}} = \mathbf{A}\dot{\mathbf{x}} \qquad (22)$$

Equation (22) represents homogeneous system of linear simultaneous differential equations of the first order; this system should be solved to find some analytical expression for the $\dot{\mathbf{x}}$ vector, and to solve such a system we need to find out the characteristic equation and the eigenvalues of the system.

2.6 Eigenvalues and the Characteristic Equation

2.6.1 Eigenvalues

Eigenvalues** are a special set of scalars (real or complex numbers) associated with a linear system of equations (the linear or linearized system of equations of the model in a matrix form – equation (22)), they are also known as the characteristic roots or proper values, or latent roots (Kreyszig, E., 1993) (Weisstein, E. W. (1999) Concise Encyclopedia of Mathematics CD-ROM).

The eigenvalues are computed as the roots of the characteristic equation.

2.6.2 The Characteristic Equation

The characteristic equation is the equation that is solved to find a matrix's eigenvalues; it is also called the characteristic polynomial. From equation (19), the matrix $\mathbf{A}$ is a matrix of a system of linear equations, if there is a vector $\mathbf{r} \neq \mathbf{0}$ such that:

$$\mathbf{Ar} = \lambda\mathbf{r} \tag{23}$$

Where: λ is a scalar value.

If equation (23) could be solved, then λ is one of the eigenvalues and $\mathbf{r}$ is its corresponding right eigenvector (called right; because the vector is multiplied to the right of matrix $\mathbf{A}$, and it will be the left eigenvector if the multiplication

** In German it is called Eigenwert, "Eigen" is a German word that means Proper, while "wert" means Root (Kreyszig, E., 1997).

is to the left of matrix $\mathbf{A}$). To compute the eigenvalues and their corresponding right eigenvectors equation (23) could be reduced to:

$$\therefore \left(\mathbf{A} - \lambda\mathbf{I}\right)\mathbf{r} = \mathbf{0}$$

Using Cramer's Rule, a system of linear equations has nontrivial solutions only if the determinant of the system vanishes, so we obtain the characteristic equation (Kreyszig, E., 1993):

$$\left|\mathbf{A} - \lambda\mathbf{I}\right| = \mathbf{0} \tag{24}$$

This equation has solutions that equal the number of rows or columns of the $\mathbf{A}$ matrix. The set of all solutions of equation(24), is the set of eigenvalues. By taking each eigenvalue and substituting in equation(23), we get its corresponding eigenvector.

2.6.3 The State Space Form Solution

The solution of equation (22) could be on the following form (Kreyszig, E., 1993):

$$\dot{\mathbf{x}} = c_1 e^{\lambda_1 (t - t_0)} \mathbf{r}_1 + c_2 e^{\lambda_2 (t - t_0)} \mathbf{r}_2 + \cdots + c_n e^{\lambda_n (t - t_0)} \mathbf{r}_{N_x}$$

or,

$$\dot{\mathbf{x}} = \sum_{i=1}^{N_x} c_i e^{\lambda_i (t - t_0)} \mathbf{r}_i \tag{25}$$

Where: c_1, c_2, … and c_{N_x} are constants, $\mathbf{r}_1$, $\mathbf{r}_2$, … and $\mathbf{r}_{N_x}$ are the right eigenvectors[††] of the system and λ_1, λ_2, … and λ_{N_x} are their corresponding eigenvalues.

The constant term c_i can be computed using the initial conditions at $t = t_0$, $\dot{\mathbf{x}} = \tilde{\dot{\mathbf{x}}}$. Note that the $\tilde{\dot{\mathbf{x}}}$ vector is very well-known at every time step –by the initial time step; the $\tilde{\dot{\mathbf{x}}}$ vector should be completely known to be able to start the simulation anyway. After that, at every new time step the vector $\tilde{\dot{\mathbf{x}}}$ is known from the pervious step–, substituting in equation (25):

$$\therefore \tilde{\dot{\mathbf{x}}} = \sum_{i=1}^{N_x} c_i e^{\lambda_i (t_0 - t_0)} \mathbf{r}_i$$

$$\therefore \tilde{\dot{\mathbf{x}}} = \sum_{i=1}^{N_x} c_i \mathbf{r}_i$$

$$\therefore \tilde{\dot{\mathbf{x}}} = c_1 \mathbf{r}_1 + c_2 \mathbf{r}_2 + \cdots + c_{N_x} \mathbf{r}_{N_x}$$

$$\therefore \tilde{\dot{\mathbf{x}}} = \mathbf{r}_1 c_1 + \mathbf{r}_2 c_2 + \cdots + \mathbf{r}_{N_x} c_{N_x}$$

The last formula expresses the matrix product of a vector containing all the c_i terms and the full right eigenvector $\mathbf{r} = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \cdots & \mathbf{r}_{N_x} \end{bmatrix}$[‡‡].

$$\therefore \tilde{\dot{\mathbf{x}}} = \mathbf{r} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_x} \end{bmatrix}$$

[††] These set of eigenvectors are should be linearly independent or their corresponding eigenvalues are different.

[‡‡] Although $\mathbf{r}$ is a lower case letter, it is used to express the right eigenvectors matrix; because this matrix is the arrangement of right eigenvectors beside each other in columns. Also $\mathbf{l}$ would be used ro express the left eigenvalues matrix.

$$\therefore \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_x} \end{bmatrix} = \mathbf{r}^{-1}\tilde{\mathbf{x}} \tag{26}$$

2.6.4 The Eigenvalues and their Corresponding Modes of Behavior

2.6.4.1The Eigenvalue as a Complex Number

The Real numbers are subfield from Complex numbers (Weisstein, E. W. (1999) Concise Encyclopedia of Mathematics CD-ROM). So that it is possible to express all eigenvalues –real, imaginary or complex– as complex numbers, but sometimes with zero imaginary part in the real numbers eigenvalue case and sometimes with zero real part in the imaginary numbers eigenvalue case.

This way all eigenvalues can be put in the form of $\sigma \pm j\,\omega$, where each of those symbols σ and ω has its effect on the behavior of the model.

For fully understanding the effect of the eigenvalue on the behavior, it would be useful to define some factors like α which is the **Damping Factor** or the **Damping Constant** where $\alpha = -\sigma$, as shown in figure 2, we can identify that the behavior of the system damps faster and goes to a steady state faster as the value of α is greater, and vise versa. This is because of τ which is the **Time Constant** of the system being inversely proportional with the damping factor.

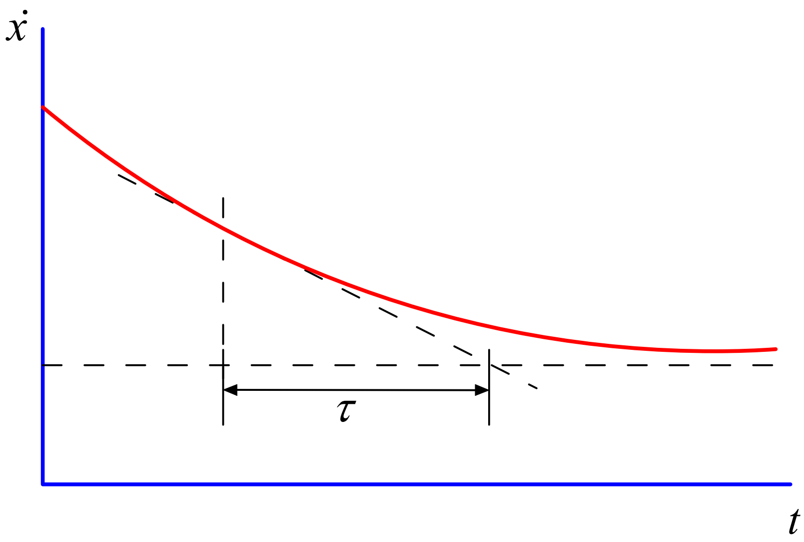


Figure 2. The System Time Constant τ

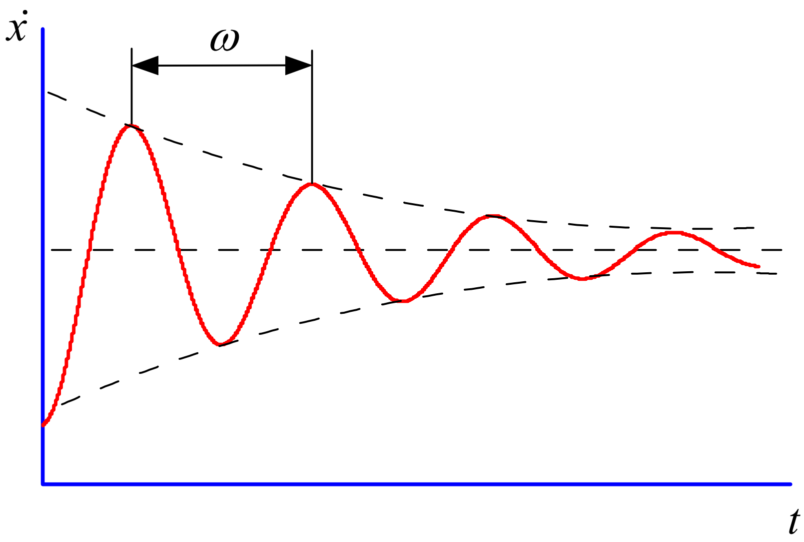


Figure 3. The System Damped Frequency ω

And

Also, because of ω which is the **Conditional Frequency** or **Damped Frequency**, which expresses the frequency of the behavior of the damped system –i.e. taking into consideration the effect of α on that system–, shown in figure 3.

For i[th] eigenvalue $\lambda_i = \sigma_i \pm j\omega_i$, by utilizing the phasor form of complex number (Edminister, Joseph A., 1972), and looking back at the analytical solution of the $\dot{\mathbf{x}}$ vector, equation (25), and noting that:

$$\because e^{\lambda_i (t - t_0)} = e^{(\sigma_i \pm j\omega_i)(t - t_0)}$$

$$\therefore e^{\lambda_i (t - t_0)} = e^{\sigma_i (t - t_0)} e^{\pm j\omega_i (t - t_0)}$$

Equation (25) could be rewritten to be:

$$\therefore \dot{\mathbf{x}} = \sum_{i=1}^{N_x} c_i e^{\sigma_i (t - t_0)} e^{\pm j\omega_i (t - t_0)} \mathbf{r}_i$$

Note that $\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{N_x} \end{bmatrix}$, also $\mathbf{r}_i = \begin{bmatrix} r_{i1} \\ \vdots \\ r_{iN_x} \end{bmatrix}$, so that we can write only the k[th] net rate:

$$\therefore \dot{x}_k = \sum_{i=1}^{N_x} c_i e^{\sigma_i (t - t_0)} e^{\pm j\omega_i (t - t_0)} r_{ki} \tag{27}$$

Where: c_i is a constant and we can multiply it to $\mathbf{r}_i$ (that contains constant terms) which gives out another vector of constants $c_i \mathbf{r}_i = \begin{bmatrix} c_i r_{i1} \\ \vdots \\ c_i r_{iN_x} \end{bmatrix} = \begin{bmatrix} c_{i1} \\ \vdots \\ c_{iN_x} \end{bmatrix}$:

$$\therefore \dot{x}_k = \sum_{i=1}^{N_x} c_{ik} e^{\sigma_i (t - t_0)} e^{\pm j\omega_i (t - t_0)} \tag{28}$$

It is easily noticed that, the behavior of one net rate is a combination of added or subtracted terms, each of those terms is related to one of the eigenvalues; which means that all eigenvalues of the system have effect on every net rate. But with a specific amplification value, this is what is meant by the constant term multiplied to each exponential term (the exponential term is the sources of dynamics in the behavior as it would be clarified).

At this point it is important to discuss the exponential term and relate it to the graph of the behavior of the net rate. The term $e^{\sigma_i (t - t_0)}$ is well-known, it is the source of exponential growth or decay –according to the sign of the σ_i – that appears in the behavior of the model. But, the other term $e^{\pm j \omega_i (t - t_0)}$ needs more investigation, this term appears only if the eigenvalue has an imaginary part, and this means that the model has another conjugate eigenvalue because complex eigenvalues come in pairs (Kreyszig, E., 1993). It might be noticed that the $\pm$ sign was used to denote the two conjugate eigenvalues.

Before going any further, let's investigate the value of c_i for both conjugate eigenvalues. Back to equation (26), and taking into consideration that $\mathbf{l} = \left(\mathbf{r}^{-1}\right)^T$, where $\mathbf{l}$ is the left eigenvector (Kreyszig, E., 1993):

$$\therefore \mathbf{l}^T = \mathbf{r}^{-1}$$

$$\therefore \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_x} \end{bmatrix} = \mathbf{l}^T \tilde{\tilde{\mathbf{x}}}$$

$$\therefore \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_x} \end{bmatrix} = \begin{bmatrix} \mathbf{l}_1 & \mathbf{l}_2 & \cdots & \mathbf{l}_{N_x} \end{bmatrix}^T \tilde{\tilde{\mathbf{x}}}$$

$$\therefore \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_x} \end{bmatrix} = \begin{bmatrix} \mathbf{l}_1^T \\ \mathbf{l}_2^T \\ \vdots \\ \mathbf{l}_{N_x}^T \end{bmatrix} \tilde{\tilde{\mathbf{x}}}$$

$$\therefore c_i = \mathbf{l}_i^T \tilde{\tilde{\mathbf{x}}}$$

If a model has two conjugate eigenvalues λ_i and λ_{i+1}, then their corresponding left eigenvectors $\mathbf{l}_i$ and $\mathbf{l}_{i+1}$ would be conjugate and their right eigenvectors $\mathbf{r}_i$ and $\mathbf{r}_{i+1}$ would be conjugate too (Kreyszig, E., 1993), consequently $\mathbf{l}_i^T$ and $\mathbf{l}_{i+1}^T$ would be conjugate. Multiplying both $\mathbf{l}_i^T$ and $\mathbf{l}_{i+1}^T$ to the same $\tilde{\mathbf{x}}$ keeps the conjugate relation, which means that both c_i and c_{i+1} would be a conjugate pair. As a result $c_i \mathbf{r}_i$ and $c_{i+1}\mathbf{r}_{i+1}$ would be a conjugate pair too, consequently c_{ik} and $c_{(i+1)k}$ are conjugate pair too.

Starting from the last deduction; for the conjugate eigenvalues, the two constants c_{ik} and $c_{(i+1)k}$ are both conjugate, i.e. we can replace $c_{(i+1)k}$ by $\bar{c}_{ik}$, which is the conjugate of c_{ik}.

The addition or subtraction of the two terms that contains the two complex conjugate eigenvalues looks like the following:

$$c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)} = e^{\sigma_i (t-t_0)} \left(c_{ik} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{-j\omega_i (t-t_0)} \right)$$

By utilizing **Euler's Formula**[§§] to the $e^{j\omega_1 (t-t_0)}$ and $e^{-j\omega_1 (t-t_0)}$ terms:

$$\therefore e^{\pm j\omega_i (t-t_0)} = \cos\left(\omega_i \left(t - t_0\right)\right) \pm j \sin\left(\omega_i \left(t - t_0\right)\right)$$

$$\therefore c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)} =$$
$$e^{\sigma_i (t-t_0)} \left(c_{ik} \cos\left(\omega_i \left(t - t_0\right)\right) + j \sin\left(\omega_i \left(t - t_0\right)\right) + \bar{c}_{ik} \cos\left(\omega_i \left(t - t_0\right)\right) - j \sin\left(\omega_i \left(t - t_0\right)\right) \right)$$

[§§] By the Swiss mathematician LEONHARD EULER:

$$e^{\pm j\theta} = \cos\theta \pm j \sin\theta$$

$$\therefore c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)} = e^{\sigma_i (t-t_0)} \left((c_{ik} + \bar{c}_{ik}) \cos\left(\omega_i (t-t_0)\right) + j (c_{ik} - \bar{c}_{ik}) \sin\left(\omega_i (t-t_0)\right) \right)$$

By replacing c_{ik} by $\frac{1}{2}(\cos\psi_i + j\sin\psi_i)$ and $\bar{c}_{ik}$ with $\frac{1}{2}(\cos\psi_i - j\sin\psi_i)$. So that the term $(c_{ik} + \bar{c}_{ik})$ equals $\cos\theta$ and the term $j(c_{ik} - \bar{c}_{ik})$ equals $-\sin\theta$:

$$\therefore c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)} = e^{\sigma_i (t-t_0)} \left(\cos\psi_i \cos\left(\omega_i (t-t_0)\right) - \sin\psi_i \sin\left(\omega_i (t-t_0)\right) \right)$$

The relation $\cos\psi \cos\left(\omega_i (t-t_0)\right) - \sin\psi \sin\left(\omega_i (t-t_0)\right)$, using Trigonometric simplification, equals $\cos\left(\omega_i (t-t_0) + \psi_i\right)$***.

$$\therefore c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)} = e^{\sigma_i (t-t_0)} \cos\left(\omega_i (t-t_0) + \psi_i\right)$$

The last equation, gives a very interesting result, a cosine wave with angular displacement ψ_i which equals $\tan^{-1}\frac{\sin\psi_i}{\cos\psi_i} = \tan^{-1}\frac{c_{ik} - \bar{c}_{ik}}{c_{ik} + \bar{c}_{ik}}$. This gives the oscillations appears in the behavior of the model. And this way we could explain the effect of the $e^{\pm j\omega_i (t-t_0)}$ term.

Before leaving this section, another two important parameters should be defined:

ζ which is the **Damping Ratio** $\zeta = \cos\theta$; where θ is the argument or phase of the eigenvalue, i.e. the angle between the eigenvalue and the positive real

---

*** $\sin(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi$ ·

line in the complex plane, and by using trigonometric simplification: $\sin\theta = \sqrt{1-\zeta^2}$ †††.

ω_n which is the **Natural Undamped Frequency**, and it is the modulus of the eigenvalue, i.e. $\omega_n = \sqrt{\sigma^2 + \omega^2}$.

††† $\sin^2\theta + \cos^2\theta = 1$
$\therefore \sin^2\theta = 1 - \cos^2\theta$
$\therefore \sin\theta = \sqrt{1-\cos^2\theta}$.

2.6.5 The Cases of Eigenvalue

2.6.5.1The first case

Shown in figure 4, the eigenvalue has zero real value and zero imaginary value:

$$\lambda_i = 0 \pm j\,0 = 0$$

Therefore $\alpha = 0$, this implies no damping over the behavior of the model.

And $\omega = 0$, this implies no oscillations.

This case is a special case; it expresses a plateau (a fixed value that is added forever to the total behavior).

The mathematical expression of the ith element (related to the ith eigenvalue) of the behavior of the kth net rate $\dot{x}_k$, is:

$$\dot{x}_{ki} = c_{ik} e^{0}$$

$$\therefore \dot{x}_{ki} = c_{ik}$$

Where: c_{ik} will stays constant forever, the behavior of this case is shown in figure 5.

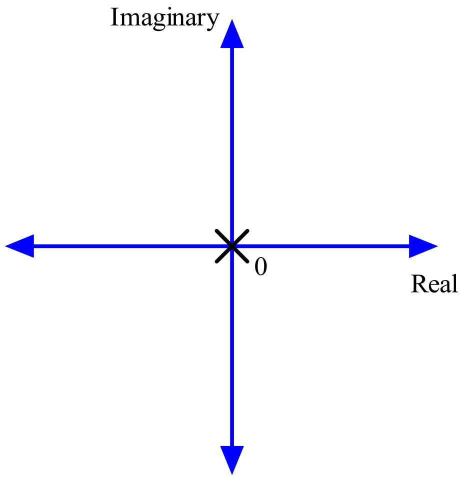


Figure 4: Zero Value Eigenvalue

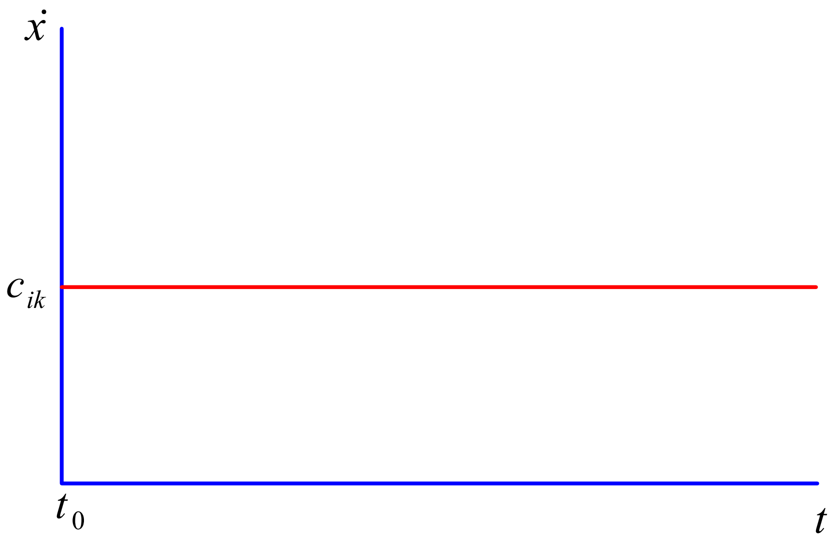


Figure 5: The Behavior Corresponding to Zero Value Eigenvalue

2.6.5.2 The second case

Shown in figure 6, the eigenvalue has positive real value and zero imaginary value:

$$\lambda_i = \sigma_i \pm j0 = \sigma_i$$

Therefore $\alpha = -\sigma_i$, this implies that damping over the behavior of the model is negative (i.e. damping is diminishing).

And $\omega = 0$, this implies no oscillations.

The mathematical expression of the i^{th} element of the behavior of the k^{th} net rate $\dot{x}_k$, is:

$$\dot{x}_{ki} = c_{ik} e^{\sigma_i (t - t_0)}$$

Where: c_{ik} remains constant forever, and the term $e^{\sigma_i (t - t_0)}$ expresses a pure exponential growth, the behavior of the model in this case is shown in figure 6.

The time constant τ or the **Doubling Time** of this behavior could be computed as follows:

$$c_{ik} \cdot e^{\sigma_i (t_2 - t_0)} = 2 \cdot c_{ik} \cdot e^{\sigma_i (t_1 - t_0)}$$

$$e^{\sigma_i (t_2 - t_0) - \sigma_i (t_1 - t_0)} = 2$$

$$\sigma_i (t_2 - t_1) = \ln(2)$$

$$t_2 - t_1 = \ln(2) / \sigma_i$$

$$\tau = \ln(2) / \sigma_i$$

Also τ could be identified from the behavior graph itself as shown in figure 6.

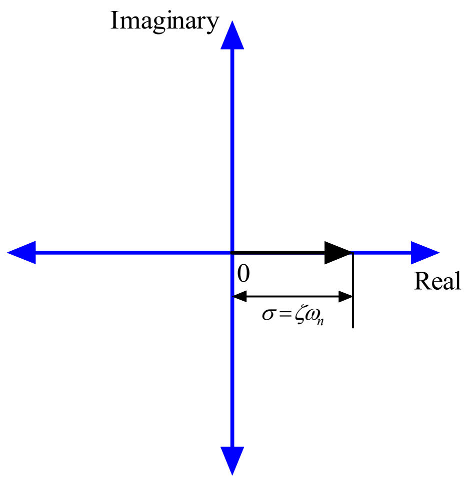


Figure 6: Positive Real Eigenvalue

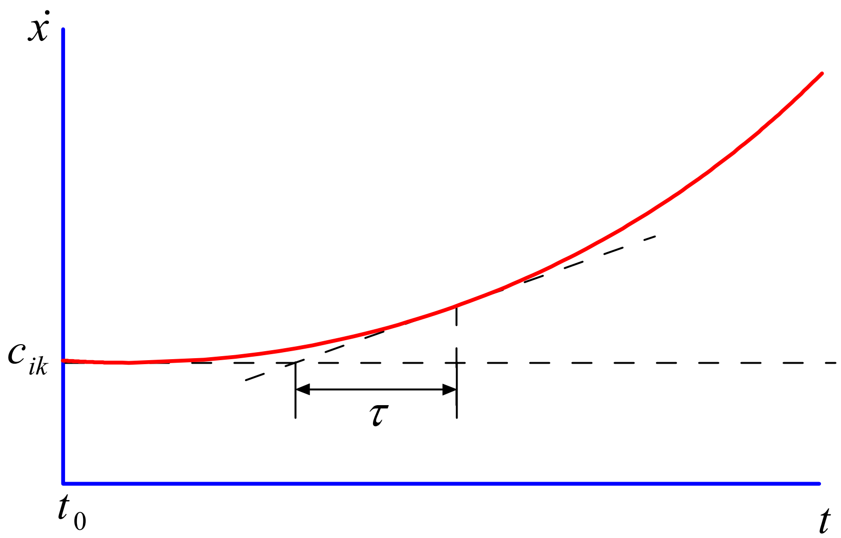


Figure 7: Mode of Behavior Corresponding to Positive Real Eigenvalue

2.6.5.3 The third case

Shown in figure 8, the eigenvalue has negative real value and zero imaginary value:

$$\lambda_i = -\sigma_i \pm j\,0 = \sigma_i$$

Therefore $\alpha = \sigma_i$, this implies that damping over the behavior of the model is positive (i.e. damping is growing).

And $\omega = 0$, this implies no oscillations.

The mathematical expression of the ith element of the behavior of the kth net rate $\dot{x}_k$, is:

$$\dot{x}_{ki} = c_{ik} e^{-\sigma_i (t - t_0)}$$

Where: c_{ik} remains constant, and the term $e^{-\sigma_i (t-t_0)}$ expresses a pure exponential decay, the behavior of the model in this case is shown in figure 9.

The time constant τ or the **Half-life Time** of this behavior could be computed as follows:

$$c_{ik} \cdot e^{-\sigma_i (t_2 - t_0)} = \frac{1}{2} \cdot c_{ik} \cdot e^{-\sigma_i (t_1 - t_0)}$$

$$e^{\sigma_i (t_2 - t_0) - \sigma_i (t_1 - t_0)} = 2$$

$$\sigma_i \left(t_2 - t_1\right) = \ln\left(2\right)$$

$$t_2 - t_1 = \ln\left(2\right) / \sigma_i$$

$$\tau = \ln\left(2\right) / \sigma_i$$

Also τ could be identified from the behavior graph itself as shown in figure 9.

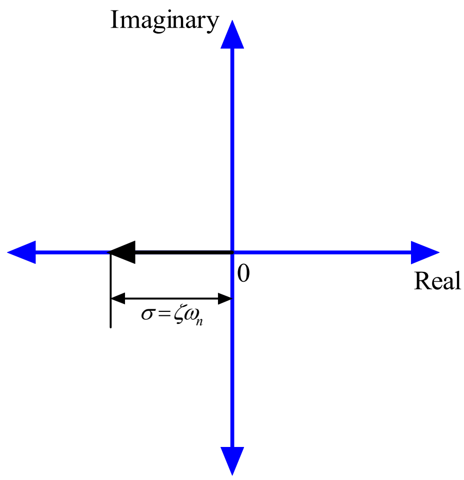


Figure 8: Negative Real Eigenvalue

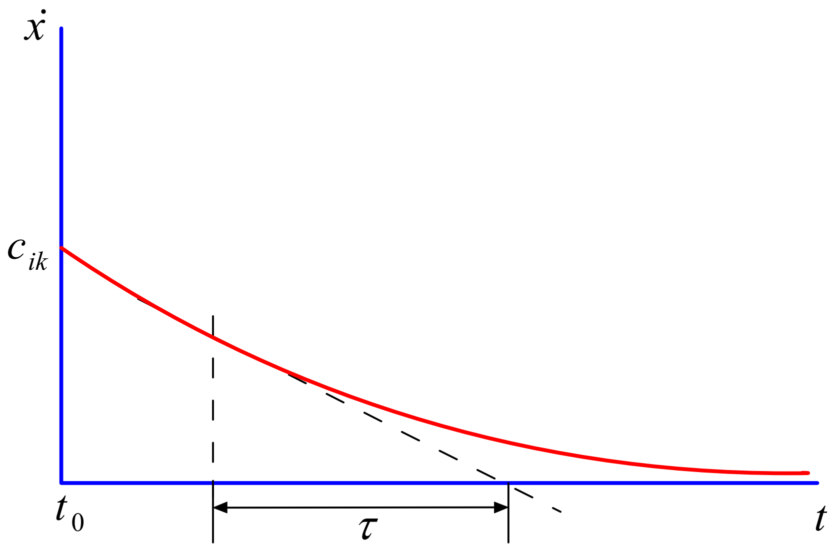


Figure 9: Mode of Behavior Corresponding to Negative Real Eigenvalue

2.6.5.4The fourth case

Shown in figure 10, the eigenvalue has an imaginary value –this means that it has another conjugate eigenvalue– The eigenvalue has zero real value and imaginary value:

$$\lambda_i = 0 \pm j\,\omega_i = \pm j\,\omega_i$$

Therefore $\alpha = 0$, this implies no damping.

And $\omega = \omega_i$, this implies that oscillations exist.

The mathematical expression of the ith element of the behavior of the kth net rate $\dot{x}_k$, is:

$$\dot{x}_{ki} = c_{ik} e^{0(t - t_0)} e^{j\,\omega_i\,(t - t_0)} + \bar{c}_{ik} e^{0(t - t_0)} e^{-j\,\omega_i\,(t - t_0)}$$

$$\therefore \dot{x}_{ki} = \cos\left(\omega_i\,(t - t_0) + \psi_i\right)$$

Where: $\psi_i = \tan^{-1} \dfrac{c_{ik} - \bar{c}_{ik}}{c_{ik} + \bar{c}_{ik}}$, and both c_{ik} and $\bar{c}_{ik}$ remain constant forever, the behavior of this case is shown in figure 11.

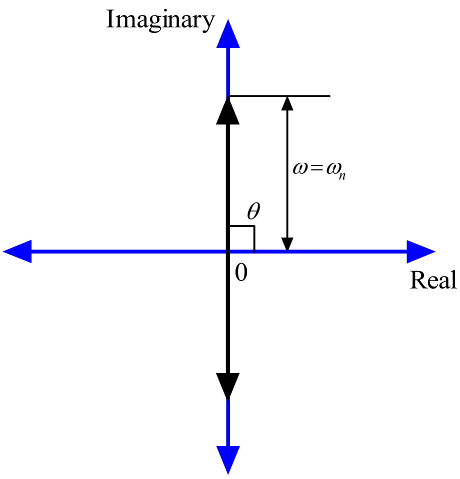


Figure 10: Pure Imaginary Eigenvalue

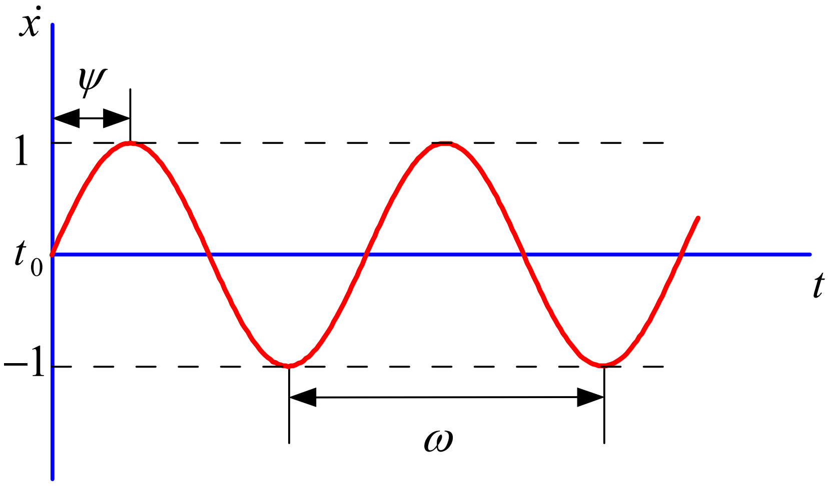


Figure 11: Mode of Behavior Corresponding to Pure Imaginary Eigenvalue

2.6.5.5 The fifth case

Shown in figure 12, the eigenvalue has an imaginary value –this is means that it has another conjugate eigenvalue– The eigenvalue has positive real value and imaginary value:

$$\lambda_i = \sigma_i \pm j\,\omega_i$$

Therefore $\alpha = -\sigma_i$, which implies that damping over the behavior of the model is negative (i.e. damping is diminishing).

And $\omega = \omega_i$, this implies that oscillations exist.

The mathematical expression of the ith element of the behavior of the kth net rate $\dot{x}_k$, is:

$$\dot{x}_{ki} = c_{ik} e^{\sigma_i (t-t_0)} e^{j\omega_i (t-t_0)} + \bar{c}_{ik} e^{\sigma_i (t-t_0)} e^{-j\omega_i (t-t_0)}$$

$$\therefore \dot{x}_{ki} = e^{\sigma_i (t-t_0)} \cos\left(\omega_i \left(t - t_0\right) + \psi_i\right)$$

Where: $\psi_i = \tan^{-1} \frac{c_{ik} - \bar{c}_{ik}}{c_{ik} + \bar{c}_{ik}}$, and both c_{ik} and $\bar{c}_{ik}$ remain constants forever, and the term $e^{\sigma_i (t-t_0)}$ expresses a pure exponential growth –the envelope of the oscillations–, the behavior of this case is shown in figure 13.

The time constant τ or the doubling time of the envelope of this behavior could be computed as in the third case:

$$\tau = \ln(2) / \sigma_i$$

Also τ could be identified from the behavior graph itself as shown in figure 13.

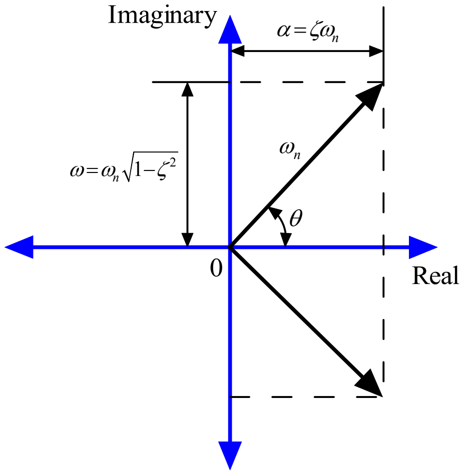


Figure 12: Imaginary Eigenvalue with Positive Real Part

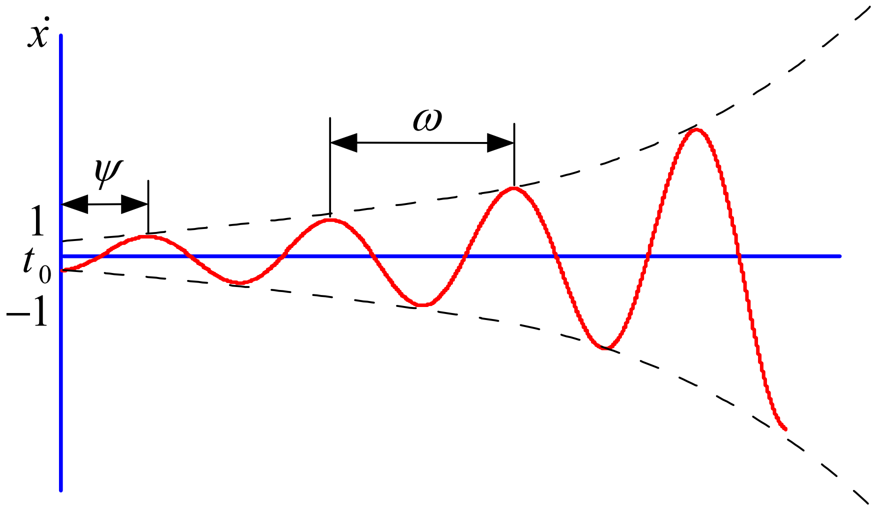


Figure 13: Mode of Behavior Corresponding to Imaginary Eigenvalue with Positive Real Part

2.6.5.6The sixth case

Shown in figure 14, the eigenvalue has an imaginary value –this is means that it has another conjugate eigenvalue– The eigenvalue has negative real value and imaginary value:

$$\lambda_i = -\sigma_i \pm j\,\omega_i$$

Therefore $\alpha = \sigma_i$, which implies that damping over the behavior of the model is positive (i.e. damping is growing).

And $\omega = \omega_i$, this implies that oscillations exist.

The mathematical expression of the i^th^ element of the behavior of the k^th^ net rate $\dot{x}_i$, is:

$$\dot{x}_{ki} = c_{ik} e^{-\sigma_i (t - t_0)} e^{j\,\omega_i (t - t_0)} + \bar{c}_{ik} e^{-\sigma_i (t - t_0)} e^{-j\,\omega_i (t - t_0)}$$

$$\therefore \dot{x}_{ki} = e^{-\sigma_i (t - t_0)} \cos\left(\omega_i \left(t - t_0\right) + \psi_i\right)$$

Where: $\psi_i = \tan^{-1} \frac{c_{ik} - \bar{c}_{ik}}{c_{ik} + \bar{c}_{ik}}$, and both c_{ik} and $\bar{c}_{ik}$ remain constants forever, and the term $e^{-\sigma_i (t - t_0)}$ expresses a pure exponential decay –the envelope of the oscillations–, the behavior of this case is shown in figure 15.

The time constant τ or the half-life time of the envelope of this behavior could be computed as in the second case:

$$\tau = \ln(2) \big/ \sigma_i$$

Also τ could be identified from the behavior graph itself as shown in figure 15.

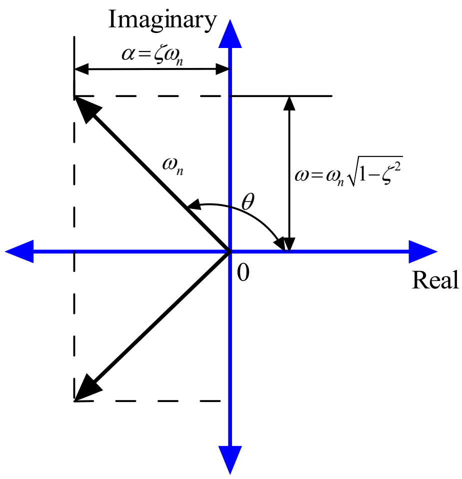


Figure 14: Imaginary Eigenvalue with Negative Real Part

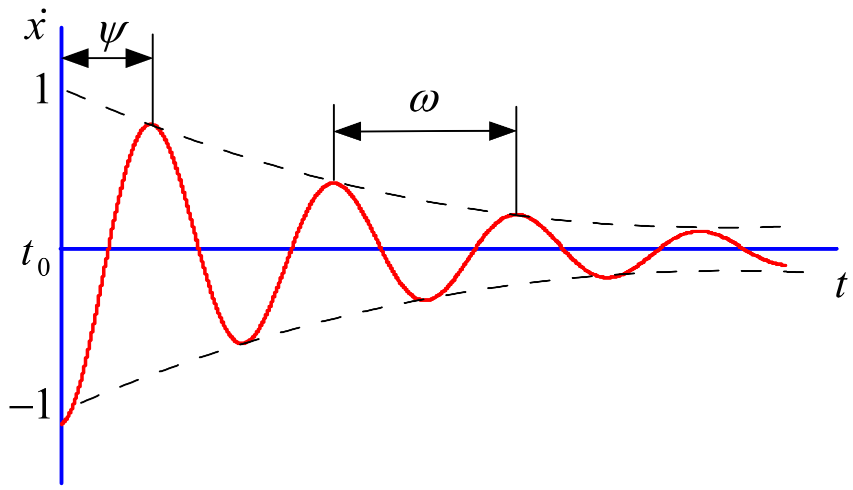


Figure 15: Mode of Behavior Corresponding to Imaginary Eigenvalue with Negative Real Part

2.7 Identifying the Dominant Eigenvalue

The total behavior of a level variable is related to the addition of the modes of behavior corresponding to the eigenvalues of the model, but with different percentages of effect. So that it is possible to specify one eigenvalue (or two conjugate pair if the eigenvalue has imaginary part) or more that affect mostly the behavior of that level variable.

The identification process of the dominant eigenvalue depends mainly on an experimental method suggested by Saleh, M. and Davidsen, P. (2000, 2001). This method depends on doing some experiments; in each experiment only one eigenvalue is allowed to affect the behavior while blocking all the other eigenvalues, then finding the dominant eigenvalue would be easy by directly comparing the results of the experiments to each other.

The last method might be improved slightly by computing the percentage of contribution of each eigenvalue on the level under study which allows arranging them according to their dominance over the level behavior, so that it would be possible to take more than only one dominant eigenvalue.

Back to equation (28), which can be rewritten to be:

$$\dot{x}_k = c_{1k} e^{\lambda_1 (t - t_0)} + c_{2k} e^{\lambda_2 (t - t_0)} + \cdots + c_{nk} e^{\lambda_n (t - t_0)}$$

or,

$$\dot{x}_k = \dot{x}_{k1} + \dot{x}_{k2} + \cdots + \dot{x}_{kn}$$

Each component in the right hand side of the last equation can be treated individually because all those components are added to or subtract from each others, so that by taking only one element:

$$\dot{x}_{ki} = c_{ik} e^{\lambda_i (t - t_0)}$$

By integrating both sides of the last equation, there are two cases:

1. $\lambda_i \neq 0$:

$$\int_{\tilde{x}_{ki}}^{x_{ki}} \dot{x}_{ki} dt = \int_{t_0}^{t} c_{ik} e^{\lambda_i (t - t_0)} dt$$

$$\int_{\tilde{x}_{ki}}^{x_{ki}} \frac{dx_{ki}}{dt} dt = \int_{t_0}^{t} c_{ik} e^{\lambda_i t} e^{-\lambda_i t_0} dt$$

$$\int_{\tilde{x}_{ki}}^{x_{ki}} dx_{ki} = \int_{t_0}^{t} c_{ik} e^{\lambda_i t} e^{-\lambda_i t_0} dt$$

$$x_{ki} \Big|_{\tilde{x}_{ki}}^{x_{ki}} = \frac{c_{ik} e^{-\lambda_i t_0}}{\lambda_i} e^{\lambda_i t} \Big|_{t_0}^{t}$$

$$x_{ki} - \tilde{x}_{ki} = \frac{c_{ik} e^{-\lambda_i t_0}}{\lambda_i} \left(e^{\lambda_i t} - e^{\lambda_i t_0} \right)$$

$$x_{ki} = \frac{c_{ik}}{\lambda_i} \left(e^{\lambda_i (t - t_0)} - 1 \right) + \tilde{x}_{ki}$$

or,

$$\delta x_{ki} = \frac{c_{ik}}{\lambda_i} \left(e^{\lambda_i \delta t} - 1 \right)$$

$$\therefore \delta x_k = \sum_{i=1}^{n} \frac{c_{ik}}{\lambda_i} \left(e^{\lambda_i \delta t} - 1 \right) \qquad (29)$$

2. $\lambda_i = 0$:

$$\int_{\tilde{x}_{ki}}^{x_{ki}} \dot{x}_{ki}\,dt = \int_{t_0}^{t} c_{ik} e^{\lambda_i (t - t_0)} dt$$

$$\int_{\tilde{x}_{ki}}^{x_{ki}} \frac{dx_{ki}}{dt}\,dt = \int_{t_0}^{t} c_{ik} e^{0(t - t_0)} dt$$

$$\int_{\tilde{x}_{ki}}^{x_{ki}} dx_{ki} = \int_{t_0}^{t} c_{ik}\,dt$$

$$x_{ki}\Big|_{\tilde{x}_{ki}}^{x_{ki}} = c_{ik}\, t\Big|_{t_0}^{t}$$

$$x_{ki} - \tilde{x}_{ki} = c_{ik}\left(t - t_0\right)$$

$$x_{ki} = c_{ik}\left(t - t_0\right) + \tilde{x}_{ki}$$

or,

$$\delta x_{ki} = c_{ik}\,\delta t$$

$$\therefore \delta x_k = \sum_{i=1}^{n} c_{ik}\,\delta t \tag{30}$$

By calculating the term δx_{ki} for each eigenvalue, it would be possible to distinguish the contribution of each eigenvalue in the behavior of x_k, where:

$$\therefore \text{contribution}_{ki} = \frac{\delta x_{ki}}{\delta x_k} \tag{31}$$

Where: contribution_{ki} is the contribution in the behavior of the kth state due to λ_i only, also we should note that the term δx_k expresses the total contribution of all eigenvalues in the value of the kth state.

By arranging the values of contribution of each eigenvalue in a descending order it would be possible to identify the dominance order of those eigenvalues.

In this context only the most dominant eigenvalue would be considered –the eigenvalue with highest contribution value–, but it is still very possible to test the effect of the second or the third dominant eigenvalue on the behavior of a state.

2.8 The Dominant Eigenvalue Elasticity

The aim of the analysis process is to identify the leverage points in the model structure. Therefore the aim of this section is to relate the dominant eigenvalue with the links of the model using what Forrester, N. (1982), has suggested to quantify that retaliation, which the eigenvalue elasticity.

2.8.1 The Eigenvalue Sensitivity

For a link that starts from a level variable x_j and ends at the net rate of another level variable x_i, the kth eigenvalue sensitivity to the gain of that link s_{kij} is defined as: the change in the kth eigenvalue due to the change in the gain of that link:

$$s_{kij} = \frac{\partial \lambda_k}{\partial a_{ij}} \tag{32}$$

or, in matrix form:

$$\mathbf{S}_k = \frac{\partial \lambda_k}{\partial \mathbf{A}} \tag{33}$$

The matrix $\mathbf{S}_k$, can be directly computed (Saleh, M., 2003):

$$\mathbf{S}_k = \mathbf{l}_k \mathbf{r}_k^{\,T} \tag{34}$$

Where: $\mathbf{l}_k$ and $\mathbf{r}_k$ are the left and right eigenvectors of the kth eigenvalue respectively.

2.8.2 The Eigenvalue Elasticity

For a link that starts from a level variable x_j and ends at the net rate of another level variable x_i, the kth eigenvalue elasticity for the gain of that link E_{kij} [‡‡‡] is defined as: the relative change in the kth eigenvalue to the relative change in the gain of that link (Saleh, M., 2003):

$$E_{kij} = \frac{\delta\lambda_k / \lambda_k}{\delta a_{ij} / a_{ij}} \tag{35}$$

$$\therefore E_{kij} = \frac{1}{\lambda_k}\frac{\delta\lambda_k}{\delta a_{ij}} a_{ij}$$

Using the definition from equation (32):

$$\therefore E_{kij} = \frac{1}{\lambda_k} s_{kij} a_{ij} \tag{36}$$

Or, in matrix form:

$$\therefore \mathbf{E}_k = \frac{1}{\lambda_k}\mathbf{S}_k \,.\!*\mathbf{A} \tag{37}$$ [§§§]

Where: $\mathbf{E}_k$ is the kth eigenvalue elasticity matrix for the compact version of the model.

[‡‡‡] The thesis normal mathematical symbolic notation would be contradicted for the elasticity matrix elements; upper case letters would be used instead of lower case letters, in order not to confuse the reader with the exponent function.
[§§§] The symbol $.*$ is the Mathworks' Matlab notation for array multiplication operator (each element from matrix to the left of the operator is multiplied by the corresponding element from the matrix to the right of the operator).

2.8.3 The Dominant Eigenvalue Elasticity Values of the Links of the Compact Model

Applying equation (37) to compute the dominant eigenvalue elasticity values of the links of the compact model enables us to relate the system behavior to the links of the compact version of the system dynamics model, i.e. the links between state variables and net rate variables.

At this point an interesting property of the eigenvalue elasticity measure – firstly noticed by Forrester, N. – should be stated: the eigenvalue elasticity values is like electric current, that is, all eigenvalue elasticity values entering a variable in the model should come out of it again (Forrester, N. B., 1982), this property was proved after that by Saleh, M. (Saleh, M., 2003) –like: Kirchoff Current Law, in Electric Circuits (Edminister, Joseph A., 1972)–.

This interesting property would greatly help in distributing the dominant eigenvalue elasticity value of the link between two variables in the compact version of the model, among the links between the same two variables, in the full version of the model.

2.8.4 The Dominant Eigenvalue Elasticity Values of the Links of the Full Model

Back to equation (18), the full gain matrix is $\left.\left[\begin{array}{c|c} \mathbf{J}_{\dot{x},x} & \mathbf{J}_{\dot{x},z} \\ \hline \mathbf{J}_{z,x} & \mathbf{J}_{z,z} \end{array}\right]\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$; for any two variables having a link between them in the model, the full gain matrix has a corresponding element that has a value equals to the gain between those two variables, taking into consideration that this element column number is the number of the variable that the link starts from, and that the element row number is the number of the variable that the link ends at, giving that the variables of the model were numbered. The other elements of the full gain matrix that are not corresponding to links in the model equal zero.

In Graph Theory; such a matrix is called a digraph (directed graph), also adjacency matrix. And various well-know exhaustive search algorithms could be applied on that digraph to find paths between two variables or to find loops, the details of those algorithms are out of this scope of this context[****].

Back to the compact gain matrix and its corresponding computed dominant eigenvalue elasticity values matrix, in the compact version of the model, only variables that has a nonzero element in the compact gain matrix, has a corresponding nonzero element in the compact dominant eigenvalue elasticity values matrix and vice versa; i.e. only variables that has gain between them has a dominant eigenvalue elasticity value, and this is normal because if there is a gain between two variables in compact model, it means that there is a direct or indirect link or links between those two variables in the full version of the model, and zero gain means that there is no link between the two variables, and of course if there is no link between the two variables, there would be no dominant eigenvalue elasticity value between them equation (36).

[****] The interested reader might refer to "Graph Theory" (Diestel, R., 2000), "Advanced Engineering Mathematics" (Kreyszig, E., 1993) and the "Digraph toolbox" (Bahar, M.; Jantzen, J., 1995)

Suppose that the k[th] eigenvalue is the dominant eigenvalue, using equation (36) enables computing the dominant eigenvalue elasticity value of the link between the level variable x_j and the net rate of the level variable x_i : E_{kij} –for simplicity it might be said: to the level variable x_i directly instead of the net rate of the level variable x_i, and it would be more simple to drop the level variable statement and directly say x_j and x_i –.

Using one of the path identification algorithms to extract paths from the full model, it would be possible to identify the direct and indirect paths that starts from x_j and ends at x_i, excluding paths that pass through other states – because the gains and the dominant eigenvalue elasticity values of those paths that pass through other states are included within their original paths and taking them into consideration within this computation leads to erroneous redundancy–. Let those identified paths to be P_{ji_1}, P_{ji_2}, … and P_{ji_N} where N is the total number of paths that starts from x_j and ends at x_i, excluding paths that pass through other states.

$$P_{ji} = \left\{P_{ji_1}, P_{ji_2}, \ldots, P_{ji_N}\right\}$$

Let the gains of those paths to be $g_{P_{ji_1}}$, $g_{P_{ji_2}}$, … and $g_{P_{ji_N}}$ respectively. And let their dominant eigenvalue elasticity values to be $E_{kP_{ji_1}}$, $E_{kP_{ji_2}}$, … and $E_{kP_{ji_N}}$ respectively. Note that the summation of the gains and the summation of the dominant eigenvalue elasticity values of those paths together equals a_{ij} and E_{kij} respectively (Saleh, M.; Davidsen, P. I., 2000).

$$a_{ij} = \sum_{P_s \in P_{ji}} g_{P_s}$$

$$E_{kij} = \sum_{P_s \in P_{ji}} E_{kP_s}$$

The gain of each individual path could be easily computed, i.e. the values of $g_{P_{ji_1}}$, $g_{P_{ji_2}}$, … and $g_{P_{ji_N}}$; by multiplying the gains of the elements g_{ℓ_r} (links) of each path individually from the full gain matrix (Kuo, B. C., 1995) and (Ogata, K., 1997).

$$g_{P_{ji_s}} = \prod_{\ell_r \in P_{ji_s}} g_{\ell_r}$$

To compute the dominant eigenvalue elasticity value of each path individually (Saleh, M.; Davidsen, P. I., 2000):

$$E_{kP_{ji_s}} = g_{P_{ji_s}} \frac{E_{kij}}{a_{ij}} \qquad (38)$$

Or, by utilizing equation (36):

$$E_{kP_{ji_s}} = g_{P_{ji_s}} \frac{s_{kij}}{\lambda_k} \qquad (39)$$

Where: $E_{kP_{ji_s}}$ is the dominant eigenvalue elasticity values for the path P_{ji_s}.

Note that, $E_{kP_{ji_s}}$ is also the dominant eigenvalue elasticity value for every element in the path P_{ji_s}.

Also it is important to note that: one link ℓ_r could be a member of more than one path in the full version of the model and each of those paths has its distinct dominant eigenvalue elasticity value; in this case its dominant eigenvalue elasticity value of that link is the summation of all dominant eigenvalue elasticity values of all paths that pass through this link (Forrester, N. B., 1982) and (Saleh, M., 2003).

$$E_{k\ell_r} = \sum_{P_{ji_s} \supset \{\ell_r\}} E_{kP_{ji_s}} \tag{40}$$

The dominant eigenvalue elasticity values for all links of the full version of the model could be computed using equation (40), i.e. compute the **Full Dominant Eigenvalue Elasticity Values Matrix**.

Also by utilizing the eigenvalue to gain sensitivity definition, we can find the dominant eigenvalue sensitivity for all links in the full the model:

$$E_{k\ell_r} = s_{k\ell_r} \frac{g_{\ell_r}}{\lambda_k}$$

From equation (39) and (40):

$$\therefore s_{k\ell_r} \frac{g_{\ell_r}}{\lambda_k} = \sum_{P_{ji_s} \supset \{\ell_r\}} E_{kP_{ji_s}}$$

$$\therefore s_{k\ell_r} \frac{g_{\ell_r}}{\lambda_k} = \sum_{P_{ji_s} \supset \{\ell_r\}} \left(g_{P_{ji_s}} \frac{E_{kij}}{a_{ij}} \right)$$

From as previously indicated; $g_{P_{ji_s}} = \prod_{\ell_v \in P_{ji_s}} g_{\ell_v}$:

$$\therefore s_{k\ell_r} \frac{g_{\ell_r}}{\lambda_k} = \sum_{P_{ji_s} \supset \{\ell_r\}} \left(\left(\prod_{\ell_v \in P_{ji_s}} g_{\ell_v} \right) \frac{E_{kij}}{a_{ij}} \right)$$

$$\therefore s_{k\ell_r} = \lambda_k \sum_{P_{ji_s} \supset \{\ell_r\}} \left(\left(\prod_{\ell_v \in P_{ji_s}} g_{\ell_v} \right) \frac{E_{kij}}{a_{ij}} \right) \frac{1}{g_{\ell_r}}$$

$$\therefore s_{k\ell_r} = \lambda_k \sum_{P_{ji_s} \supset \{\ell_r\}} \left(\left(\frac{1}{g_{\ell_r}} \prod_{\ell_v \in P_{ji_s}} g_{\ell_v} \right) \frac{E_{kij}}{a_{ij}} \right)$$

The term $\frac{1}{g_{\ell_r}} \prod_{\ell_v \in P_{ji_s}} g_{\ell_v}$, can be simplified to $\prod_{\ell_v \in P_{ji_s} - \{\ell_r\}} g_{\ell_v}$

$$\therefore s_{k\ell_r} = \lambda_k \sum_{P_{ji_s} \supset \{\ell_r\}} \left(\left(\prod_{\ell_v \in P_{ji_s} - \{\ell_r\}} g_{\ell_v} \right) \frac{E_{kij}}{a_{ij}} \right) \quad (41)$$

Back to the yeast cells example, the matrix $\mathbf{A}$ has three nonzero elements from $Alcohol$ (column 1) to $Cells$ (row 2), from $Cells$ to $Alcohol$ and from $Cells$ to $Cells$.

Let's take the link from $Alcohol$ to $Cells$, its gain equals:

$-1/150 * Cells - 1/30 * Cells * \exp(Alcohol - 11)$.

Note that: at each time step through simulation period, the matrix $\mathbf{A}$ elements will be of specific numerical values. And easily the $\mathbf{E}$ matrix elements could be numerically computed using equation (36). So that, let's assume that this numerical evaluation is done and the last gain expression is in numerical form and let's suppose that it equals a_{21} and kth eigenvalue is the dominant one, so that its dominant eigenvalue elasticity value numerically equals E_{k21}.

By applying an exhaustive search algorithm on the full gain matrix of the model, to find all possible paths from $Alcohol$ to $Cells$, the path from $Alcohol$ to $Cells$ contains two different paths:

$P_1 : Alcohol \rightarrow Eff\ \ Alcohol\ \ On\ \ Births \rightarrow Birth \rightarrow Cells$
$P_2 : Alcohol \rightarrow Eff\ \ Alcohol\ \ On\ \ Deaths \rightarrow Deaths \rightarrow Cells$

Easily the gain of each path could be computed, by multiplying gains of its elements. Let's suppose that the gains of those paths are g_{P_1} and g_{P_2} for P_1 and P_2 respectively.

By utilizing equation (38), the dominant eigenvalue elasticity for each path individually is:

$$E_{kP_1} = g_{P_1} \cdot \frac{E_{k\,21}}{a_{21}}$$

$$E_{kP_2} = g_{P_2} \cdot \frac{E_{k\,21}}{a_{21}}$$

Where: E_{kP_1} and E_{kP_2} are the dominant eigenvalue elasticity values for P_1 and P_2 respectively.

Note that, E_{kP_1} is the dominant eigenvalue elasticity values for every element in P_1, and the same for E_{kP_2} and the elements of P_2.

2.8.5 The Dominant Eigenvalue Elasticity for the Inputs

To get the full benefit of the knowledge of the dominant eigenvalue elasticity values of the system dynamics model links, it should be possible to change the gains of those links to be able to change the dominant eigenvalue, i.e. changing the gain of a link so that the eigenvalue and of course the system behavior would change in a desirable way.

As stated before the model inputs or constants or parameters are the controllable part of the model, so that it should be possible to compute the dominant eigenvalue elasticity values for the model inputs. Utilizing the following relation (Saleh, M., 2003):

$$E_{ku_q} = \frac{\delta\lambda_k / \lambda_k}{\delta u_i / u_q} \tag{42}$$

Where: E_{ku_q} is the dominant eigenvalue elasticity value for the input u_q.

By rearranging the terms of equation (42):

$$\therefore E_{ku_q} = \frac{1}{\lambda_k} \frac{\partial \lambda_k}{\partial u_q} u_q$$

Applying the Chain Rule (Kreyszig, E., 1993):

$$E_{ku_q} = \frac{1}{\lambda_k} \left(\sum_{r=1}^{N_\ell} \frac{\partial \lambda_k}{\partial g_{\ell_r}} \frac{\partial g_{\ell_r}}{\partial u_q} \right) u_q$$

Where: N_ℓ is the number of all links in the full model.

But from the definition of the eigenvalue to gain sensitivity $s_{k\ell_r} = \frac{\partial \lambda_k}{\partial g_{\ell_r}}$:

$$\therefore E_{ku_q} = \frac{1}{\lambda_k} \left(\sum_{r=1}^{N_\ell} s_{k\ell_r} \frac{\partial g_{\ell_r}}{\partial u_q} \right) u_q \tag{43}$$

Where: $s_{k\ell_r}$ can be computed directly using equation (41).

2.8.6 The Dominant Eigenvalue Elasticity for the Loops

The loops of a model are the most meaningful building blocks. As stated before, the full gain matrix is a digraph, and could be searched using search algorithms; to find paths between two variables, and loops by identifying paths that starts from a variable and ends at the same variable.

By identifying loops, their gains and dominant eigenvalue elasticity values could be identified.

2.8.6.1Identifying Loops in the Model

Kampmann, C. E. (1996) suggested a binary matrix that relates the links with the loops:

$$\begin{bmatrix} \ell_1 \\ \ell_2 \\ \vdots \\ \ell_{N_\ell} \end{bmatrix} = \mathbf{C} \begin{bmatrix} \kappa_1 \\ \kappa_2 \\ \vdots \\ \kappa_{N_\kappa} \end{bmatrix} \tag{44}$$

Where: κ_i expresses the ith loop, ℓ_j expresses the jth link. N_κ and N_ℓ are the number of all loops and all links in the model respectively. The matrix **C** would be a non-square binary matrix:

$$\mathbf{C} = \left[c_{ij} \right]$$

Where: c_{ij} = 1 if the link ℓ_j is a component in loop κ_i , 0 otherwise.

Back to the yeast cells model, and by applying a search algorithm to the model, the following four loops would be identified:

κ_1 : *Cells* → *Births* → *Cells*
κ_2 : *Cells* → *Deaths* → *Cells*
κ_3 : *Alcohol* → *Eff Alcohol On Births* → *Births* → *Cells* → *Alcohol Generation* → *Alcohol*
κ_4 : *Alcohol* → *Eff Alcohol On Deaths* → *Deaths* → *Cells* → *Alcohol Generation* → *Alcohol*

$$\therefore \kappa = \begin{bmatrix} \kappa_1 \\ \kappa_2 \\ \kappa_3 \\ \kappa_4 \end{bmatrix}$$

Also,

$\ell_1 : Alcohol \rightarrow Eff\ \ Alcohol\ \ On\ \ Births$
$\ell_2 : Alcohol \rightarrow Eff\ \ Alcohol\ \ On\ \ Deaths$
$\ell_3 : Cells \rightarrow Alcohol\ \ Generation$
$\ell_4 : Cells \rightarrow Births$
$\ell_5 : Cells \rightarrow Deaths$
$\ell_6 : Eff\ \ Alcohol\ \ On\ \ Births \rightarrow Births$
$\ell_7 : Eff\ \ Alcohol\ \ On\ \ Deaths \rightarrow Deaths$
$\ell_8 : Alcohol\ \ Generation \rightarrow Alcohol$
$\ell_9 : Births \rightarrow Cells$
$\ell_{10} : Deaths \rightarrow Cells$

$$\therefore \ell = \begin{bmatrix} \ell_1 \\ \ell_2 \\ \ell_3 \\ \ell_4 \\ \ell_5 \\ \ell_6 \\ \ell_7 \\ \ell_8 \\ \ell_9 \\ \ell_{10} \end{bmatrix}$$

$$\therefore \mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

2.8.6.2Linearly Independent Loops and their Dominant Eigenvalue Elasticity Values

As stated before, Forrester, N. B. (1982) has discovered the similarity between eigenvalue elasticity and electric current. Also Kampmann, C. E. (1996) suggested equation (44), but he stated that solving this equation in its form wouldn't be possible, also he added (from the graph theory) that for this equation to be solvable; the set of all loops should be replaced by a smaller set of loops that is their number equals to (*total number of links – total number of variables* + 1). And that this is exactly the number of any selected linearly independent loop set.

So that in equation (44), by replacing the $\mathbf{C}$ with another smaller matrix $\mathbf{C}_r$, to relate the eigenvalue elasticity values of links with that of a linearly independent loop set of loops.

$$\begin{bmatrix} E_{k\ell_1} \\ E_{k\ell_2} \\ \vdots \\ E_{k\ell_{N_\ell}} \end{bmatrix} = \mathbf{C}_r \begin{bmatrix} E_{k\kappa_1} \\ E_{k\kappa_2} \\ \vdots \\ E_{k\kappa_{N_\kappa}} \end{bmatrix} \qquad (45)$$

Where: $E_{k\kappa_i}$ and $E_{k\ell_j}$ express the dominant eigenvalue elasticity values of the ith loop and the jth link respectively.

Or, in matrix form:

$$\mathbf{E}_{k\ell} = \mathbf{C}_r \mathbf{E}_{k\kappa} \qquad (46)$$

Equation (46) could be easily solved for $\mathbf{E}_{k\kappa}$ using least squares solution, but the real problem is how to select the matrix $\mathbf{C}_r$ from the rows of the matrix $\mathbf{C}$, in other words; how to select the linearly independent loops set.

> *"The rank of a matrix A is the maximum number of linearly independent columns of A; or it is the order of the largest nonsingular matrix contained in A."*
>
> *–Kuo, B. C. (1995).*

This makes it easy to find out the $\mathbf{C}_r$, knowing that it is not unique for the model, i.e. there could be more than one linearly independent loops set (Kampmann, C. E., 1996), also Kampmann suggested that the user should select the most significant set for his model from the user's point of view.

Back to the yeast cells model, by computing the rank of the matrix $\mathbf{C}$; it equals 4. This is while; the total number of links equals 10 and the total number of variables equals 7, so that the number of linearly independent loops set, should be 10 – 7 + 1 = 4, which is the same result of the rank. Which also means that all loops in the model are linearly independent, i.e. $\mathbf{C}_r = \mathbf{C}$.

Easily by substituting in equation (45) and solving it for the eigenvalue elasticity values of the loops, this system of equations is over determined system; i.e. the number of equations is greater than the number of unknowns. But it is still a consistent system that could be solved and give exact values to the unknowns.

Chapter 3

The Analysis Package: Computer Implementation

3.1 Introduction

This chapter focuses on the implementation of the functions of the Analysis package using the programming language of Mathworks Matlab mathematical package. The Analysis package consists of many functions that aim at applying the eigenvalue analysis steps on system dynamics models in Powersim constructor text file format. In fact, the Analysis package functions don't deal directly with the model file; instead they take their intputs from the Simulation package[††††]. The following sections of this chapter present the functions of the package.

Section 3.2 3.2presents the *analysis* function; which is the backbone function of the Analysis package that calls all the other functions.

Section 3.3 presents the *extractModelObjects* function; which extracts information from inputs that come from the Simulation package to the Analysis package.

Section 3.4 presents the *computeSystemJacobians* function; which computes in symbolic form, two of the most important matrices needed through the usage of the package.

Section 3.5 presents the *findIndependentCycles* function; which finds sets of loops and does their calculations.

[††††] The Simulation package is a Powersim model text file parser and simulation package implemented by Bahaa El-Din Ali Abdel-Aleem as a technical part of his master thesis – Bergen University.

Section 3.6 presents the *findDominantEigenvalue* function; which identifies the dominant eigenvalue.

Sections 3.7, 3.8 and 3.9 present the *computeLinkElasticity*, *computeInputElasticity*, *computeIndependentCycleElasticity* functions; which compute dominant eigenvalue elasticity values for links, inputs and linearly independent loops respectively.

Section 3.10 presents the *printOutputs* function; which prints all outputs of the package.

Sections 3.11, 3.12 and 3.13 present the *jac*, *differentiate*, *differentiateGraph* functions respectively; which are related to finding different differentiations inside the package.

In order to make the explanation of the functions as clear and sorted as possible; the *analysis* function –although it is the main function in the Analysis package that calls all the other functions– is treated through explanation as a normal function, and all functions would be explained in the order of their call.

Shown in figure 16; the context level diagram of both simulation and analysis packages together, which declares the relation among the main entities and both packages as a single process. While in figure 17; the data flow diagram (DFD) level zero of both packages, which declares the relation and data flow between them in details. For more information about the data stores (variables); the reader should refer to the appendices.

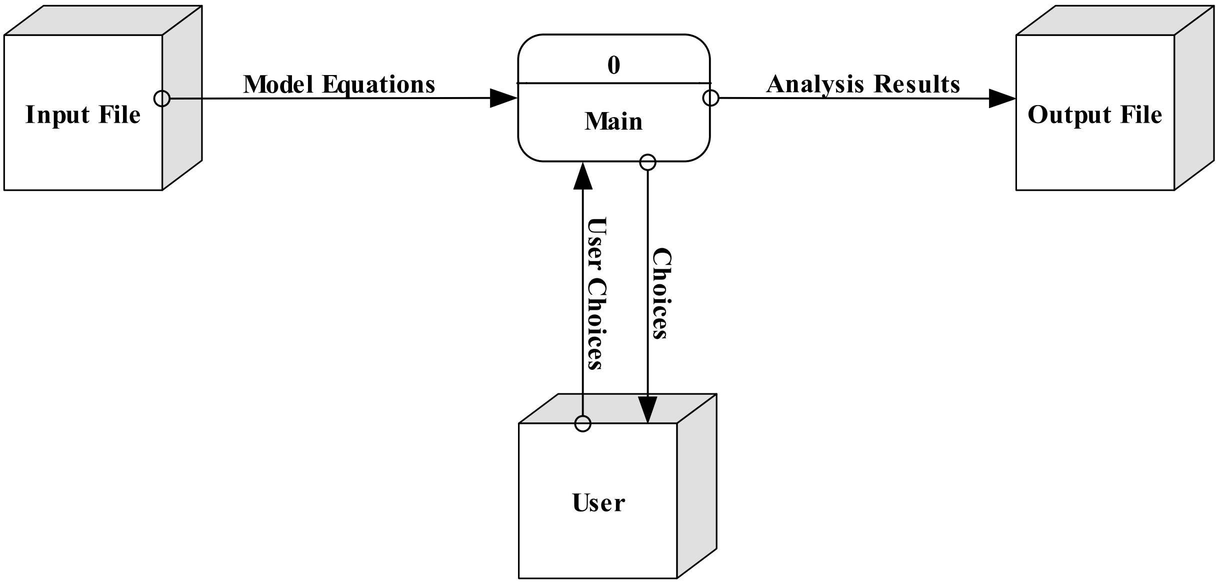


Figure 16: The Context Level Diagram of Both Simulation and Analysis Packages Together

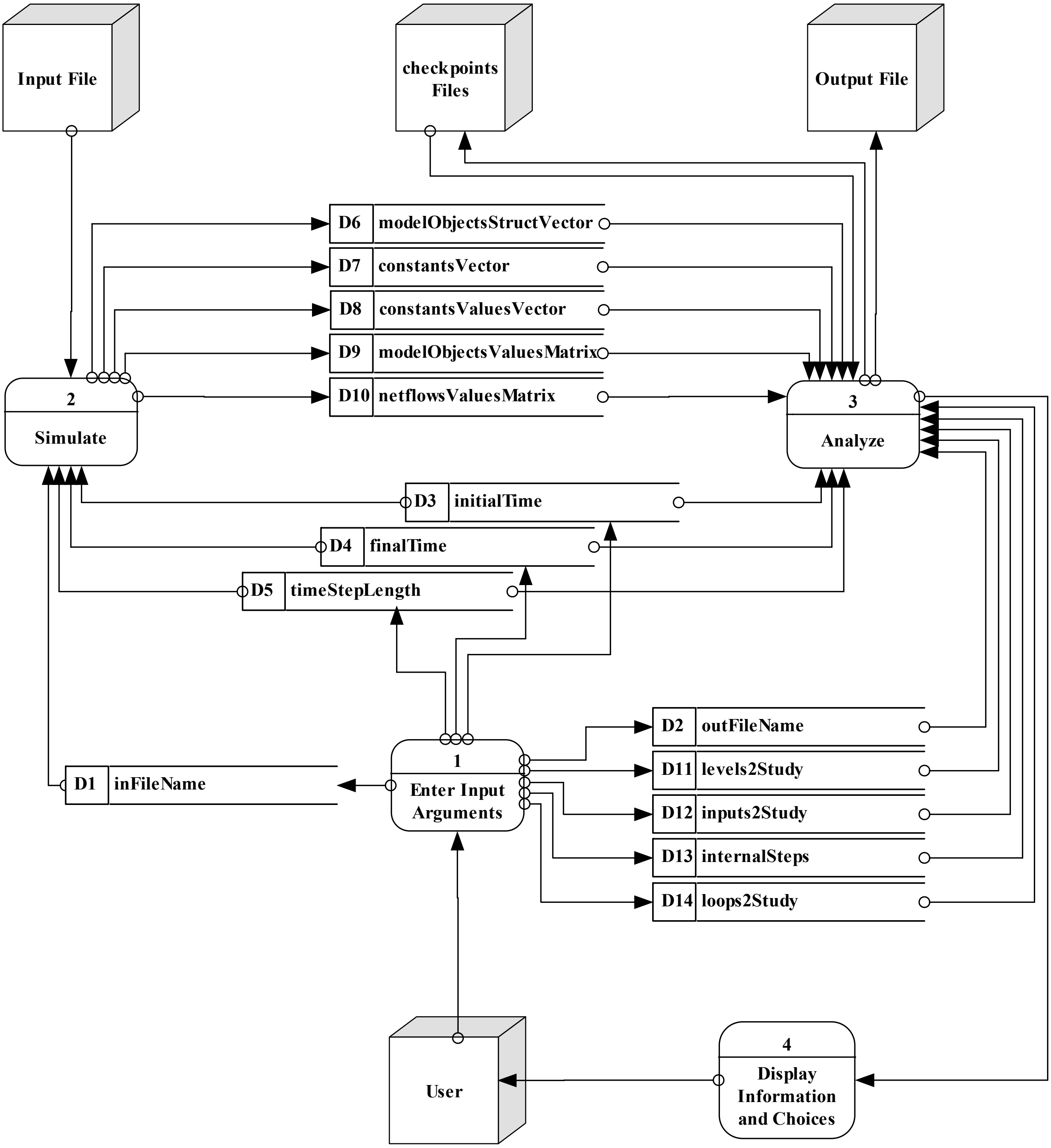


Figure 17: The Data Flow Diagram (DFD) Level Zero: Simulation and Analysis Packages

3.2 The analysis Function

The *analysis* function is the backbone of the Analysis package; it carries out a big part of the eigenvalue analysis process as well as it calls all other functions of the Analysis package.

3.2.1 Extracting Objects of the Model

The function calls another function named *extractModelObjects* to compute the following scalars and vectors:

- *numLevels*: number of levels
- *numAuxiliaries*: number of auxiliaries
- *modelObjectsNamesVector*: vector of names of all level and all auxiliary variables together
- *modelObjectsEquationsVector*: vector of equations of all net-flows and all auxiliary variables together

For more information about the internals of this process, the reader should refer to the section of *extractModelObjects* function.

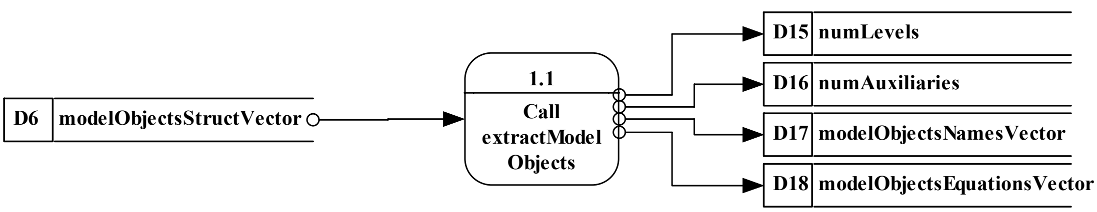


Figure 18: The DFD Level One: Extracting Objects of the Model

```
% Extracting Objects of the Model
[ numLevels , ...
  numAuxiliaries , ...
  modelObjectsNamesVector , ...
  modelObjectsEquationsVector ] = extractModelObjects(
modelObjectsStructVector );
```

3.2.2 Emptying and Initializing Checkpoints

The *Analysis* package generates eight checkpoint files; these checkpoint files enable the user to check the accuracy of the internal calculations performed by the package concerning his/her model through the analysis process.

The following code listing contains the process of emptying and initializing of only one checkpoint file. The other checkpoint files have the same lines of code to perform file emptying and initializing, they differ just in the text printed inside each file.

The function performs the following steps:

- Empties the file by opening it in the write mode; this creates the file if it doesn't exist or overwrites it otherwise
- Writes the initialization lines (specific for each checkpoint file) into the empty file
- Closes the file

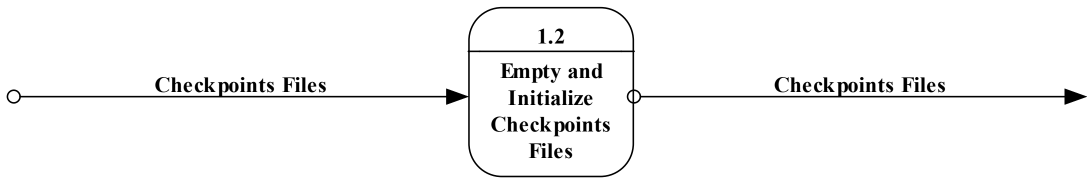


Figure 19: The DFD Level One: Emptying and Initializing Checkpoints

```
% Emptying and Initializing Checkpoint(0)
fid = fopen( [ 'checkpoint_0.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"findDominantEigenvalue.m" at the end of the file,' sprintf( '\n' ) ]
);
fwrite( fid , [ 'it contains the following:' sprintf( '\n' ) ] );
fwrite( fid , [ 'it computes the error (E) and percentage error (PE)'
sprintf( '\n' ) ] );
fwrite( fid , [ 'between the absolute value of:' sprintf( '\n' ) ] );
fwrite( fid , [ 'next time step State Vector X(t+1),' sprintf( '\n' )
```

```
] );
fwrite( fid , [ 'the one comes from simulation' sprintf( '\n' ) ] );
fwrite( fid , [ 'and the computed one from the (alpha / lambda) *
exp(lambda * dt) equations ...' sprintf( '\n\n' ) ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
    fwrite( fid , [ 'E (X' num2str( I ) ');PE (X' num2str( I ) ');' ]
);
end
fwrite( fid , [ sprintf( '\n\n' ) ] );
fclose( fid );
```

3.2.3 Calculating Number of Time Steps

The function calculates the number of time steps *numTimeSteps* using the values of the following variables:

- *initialTime*: initial time
- *finalTime*: final time
- *timeStepLength*: length of the time step

The function performs this calculation according to the following equation:

$$numTimeSteps = \frac{finalTime - initialTime}{timeStepLength} + 1$$

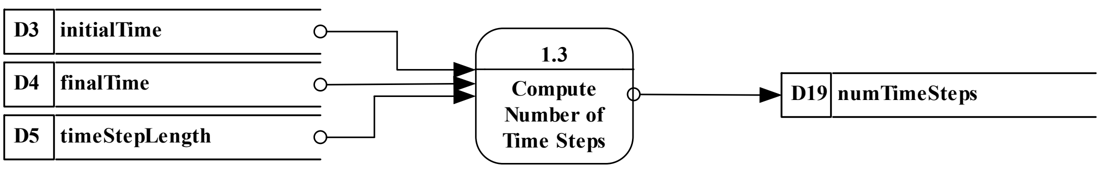


Figure 20: The DFD Level One: Calculating Number of Time Steps

```
% Calculating Number of Time Steps
numTimeSteps = ( ( finalTime - initialTime ) / timeStepLength ) + 1;
```

3.2.4 User-Interaction: Selecting Level to Study

The function needs user-interaction to make decisions concerning the flow of its execution flow.

This is the first user-interaction; the function needs the user to decide the level variable he/she wants to study its behavior.

The function performs the following steps:

- Goes into an endless while-loop
- Prints all levels in a numbered style using a for-loop
- Asks the user to choose the level he/she wants to study its behavior by entering its corresponding number
- Saves the user input into a variable named *levels2Study*
- Checks *levels2Study* and ends the while-loop if *levels2Study* is of an appropriate value, or keeps looping inside the while-loop

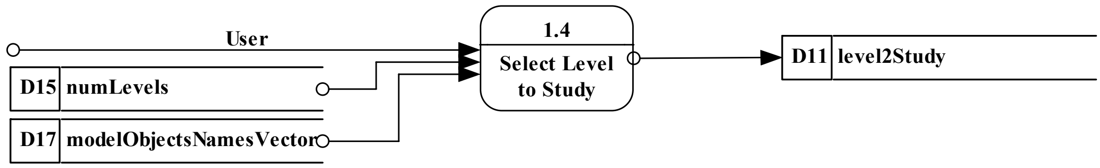


Figure 21: The DFD Level One: Selecting Level to Study

```
% User-Interaction: Selecting Level to Study
endLoop = true;
while ( endLoop )
    for I = 1 : numLevels,
        disp( [ int2str( I ) ' - ' char( modelObjectsNamesVector( I )
) ] );
    end
    levels2Study = input( [ 'Enter the number of the level, you are
intersted' sprintf( '\n' ) 'in studying  (ex.: 2):' sprintf( '\t' ) ]
);
    if length( levels2Study ) ~= 1 | levels2Study > numLevels |
levels2Study < 1,
        disp( 'Wrong Input(s), try again ...' );
    else
        endLoop = false;
    end
end
```

3.2.5 User-interaction: Selecting Inputs to Study

Exactly using the same way in the previous section, the function needs the user to decide his/her set of inputs out of the set of all constants in the model.

The function performs the following steps:

- Goes into an endless while-loop
- Prints all constants in a numbered style using a for-loop
- Asks the user to choose the set of constants he/she wants to consider as inputs by entering their corresponding numbers in a vector form
- Saves the user input into a variable named *inputs2Study*
- Checks *inputs2Study* and ends the while-loop if *inputs2Study* is appropriate, or keeps looping inside the while-loop

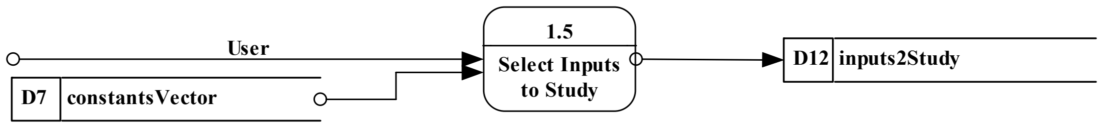


Figure 22: The DFD Level One: Selecting Inputs to Study

```
% User-Interaction: Selecting Inputs to Study
endLoop = true;
while ( endLoop )
    for I = 1 : length( constantsVector ),
        disp( [ int2str( I ) ' - ' char( constantsVector( I ) ) ] );
    end
    inputs2Study = input( [ 'Enter the number of constants, you would
like to' sprintf( '\n' ) 'consider as inputs  (ex.: 30 or [ 1,2,3 ]
or [ 1:50 ] ):' sprintf( '\t' ) ] );
    if isempty( inputs2Study ),
        inputs2Study = [ 1 : length( constantsVector ) ];
        endLoop = false;
    elseif inputs2Study > length( constantsVector ) | max(
inputs2Study ) > length( constantsVector ) | min(inputs2Study) < 1,
        disp( 'Wrong Input(s), try again ...' );
    else
        endLoop = false;
    end
```

```
end
```

3.2.6 Suggesting Time Steps to Apply Eigenvalue Analysis to

Applying eigenvalue analysis to all time steps would consume a lot of time according to the computer processing power and speed, so that the function helps the user by dividing the behavior of the user-selected level into spans according to the Behavior Pattern Index (BPI), and choosing points in the middle of each of these span.

The function performs the following steps:

- Computes *curvature*: the curvature using the following equation:

$$curvature(t) = \frac{netflowsValuesMatrix(t) - netflowsValuesMatrix(t-1)}{timeStepLength}$$

- Computes *BPI*: the BPI using the following equation (Saleh, M.; Davidsen, P. I., 2000):

$$BPI(t) = \operatorname{sgn}\left(\frac{curvature(t)}{netflowsValuesMatrix(t, levels2Study)}\right)$$

- Conditions *BPI* to replace zeros and non-numeric values (division-by-zero results) by suitable values
- Combines repeated similar *BPI* into spans and saves them into a vector named *BPI_spans*
- Chooses a step in the middle of each BPI span and saving them into a vector named *suggestedInternalStep*

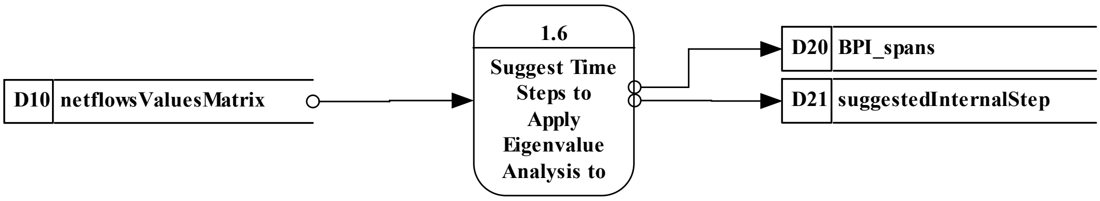


Figure 23: The DFD Level One: Suggesting Time Steps to Apply Eigenvalue Analysis to

```
% Suggesting Time Steps to Apply Eigenvalue Analysis to
curvature = zeros( size( netflowsValuesMatrix( : , levels2Study ) )
);
curvature( 2 : end ) = diff( netflowsValuesMatrix( : , levels2Study )
) / timeStepLength;
BPI = sign( curvature( : ) ./ netflowsValuesMatrix( : , levels2Study
) );
if isnan( BPI( end ) ),
    endLoop = true;
    J = [ length( BPI ) ];
    I = length( BPI ) - 1;
    while( endLoop ),
        if ~isnan( BPI( I ) ),
            endLoop = false;
        else
            J = [ J , I ];
            I = I - 1;
        end
    end
    BPI( J ) = BPI( I );
end
endLoop = true;
J_All = find( isnan( BPI ) );
for K = J_All.',
    if isnan( BPI( K ) ),
        J = K;
        I = K + 1;
        while( endLoop ),
            if ~isnan( BPI( I ) ),
                endLoop = false;
            else
                J = [ J , I ];
                I = I + 1;
            end
        end
        BPI( J ) = BPI( I );
    end
end
```

```
BPI_spans = diff( BPI );
BPI_spans=[ 1 ; find( abs( BPI_spans( : ) ) == 2 ) ; numTimeSteps ];
suggestedInternalStep = BPI_spans + [ round( diff( BPI_spans ) / 2 )
; 0 ];
suggestedInternalStep( end ) = [];
```

3.2.7 Plotting the Selected Level to Study

The function plots the behavior of the user-selected level divided into the BPI spans computed in the previous section.

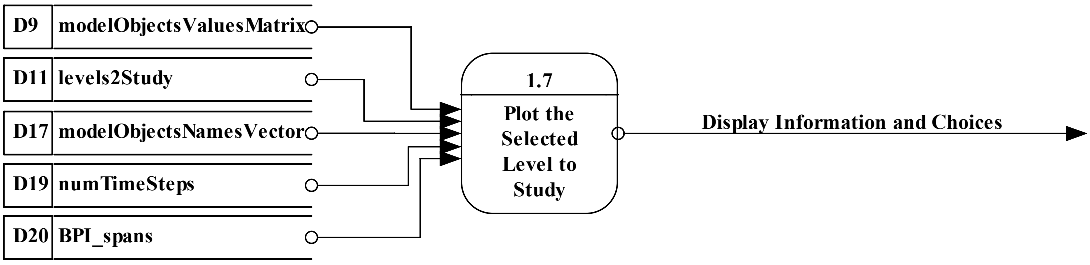


Figure 24: The DFD Level One: Plotting Selected Level to Study

```
% Plotting Selected Level to Study
plot( modelObjectsValuesMatrix( 1 : numTimeSteps , levels2Study ) ,
'LineWidth' , 2 );
set( gca , 'XTick' , BPI_spans );
set( gca , 'XTickLabel' , { num2str( round( ( BPI_spans - 1 ) *
timeStepLength * 10 ) / 10 ) } );
set( gca , 'XGrid' , 'on' );
axis tight;
xlabel( 'time' );
title( char( modelObjectsNamesVector( levels2Study ) ) );
```

3.2.8 User-interaction: Selecting Time Steps to Apply Eigenvalue Analysis to

Although the function has computed the most appropriate time steps to apply the eigenvalue analysis to, the user is free to select the time steps he/she would like. So that the function prints the suggested time steps and waits for user input, then saves into a vector named *internalSteps*.

The function performs the following steps:

- Goes into an endless while-loop
- Prints the model time range and the suggested time steps
- Asks the user to select the time steps he/she wants to apply the eigenvalue analysis to by entering them in a vector form
- Saves the user input into a vector named *internalSteps*
- Checks *internalSteps* and ends the while-loop if *internalSteps* is appropriate, or still looping inside the while-loop

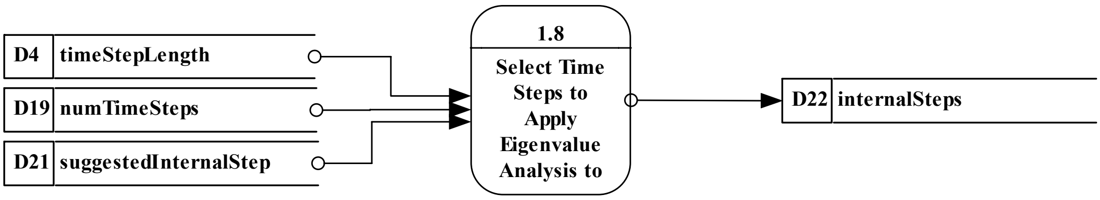


Figure 25: The DFD Level One: Selecting Time Steps to Apply Eigenvalue Analysis to

```
% User-interaction: Selecting Time Steps to Apply Eigenvalue Analysis
to
endLoop = true;
while ( endLoop )
    disp( [ 'Time Steps range is from 1 to ' int2str( numTimeSteps )
] );
    disp( [ 'Corresponding to Time Instants range from' num2str(
initialTime ) ' to ' num2str( finalTime ) ] );
    disp( [ 'it is suggested to do analysis at the following time
steps: ' sprintf( '\n' ) int2str( suggestedInternalStep.' ) ] );
    disp( [ 'Corresponding to following time instants: ' sprintf(
'\n' ) num2str( [ ( suggestedInternalStep - 1 ) * timeStepLength ].'
) ] );
    internalSteps = input( [ 'Enter the time steps, you are
intersted' sprintf( '\n' ) 'in studying  (ex.: 30 or [ 1,2,3 ] or [
1:50 ] ):' sprintf( '\t' ) ] );
    if isempty( internalSteps ),
        internalSteps = [ 1 : numTimeSteps ];
        endLoop = false;
    elseif max( internalSteps ) > numTimeSteps | min( internalSteps )
< 1,
        disp( 'Wrong Input(s), try again ...' );
    else
        endLoop = false;
```

```
    end
end
```

3.2.9 Computing Adjacency Matrix and Jacobians of the Model

The function calls another function named *computeSystemJacobians* to compute the following matrices:

- *symbolicFullGainMatrix*: the full gain matrix of the model in symbolic form
- *symbolicLinkGain2InputJacobianMatrix*: the Jacobian of the links' gains to the inputs of the model in symbolic form
- *modelAdjacencyMatrix*: the model adjacency matrix or model digraph
- *modelAdjacencyMatrix2EdgesMatrix*: a dictionary matrix to translate a link into its start and end variables, and vice versa

For more information about the internals of this process, the reader should refer to the section of *computeSystemJacobians* function.

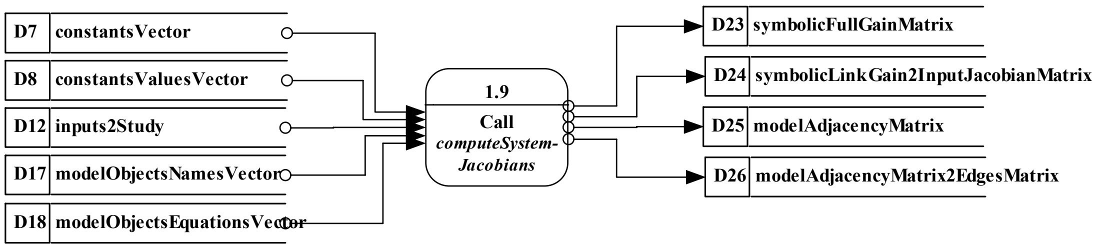


Figure 26: The DFD Level One: Computing Adjacency Matrix and Jacobians of the Model

```
% Computing Adjacency Matrix and Jacobians of the Model
[ symbolicFullGainMatrix , ...
  symbolicLinkGain2InputJacobianMatrix , ...
  modelAdjacencyMatrix , ...
  modelAdjacencyMatrix2EdgesMatrix ] = computeSystemJacobians(
modelObjectsNamesVector , modelObjectsEquationsVector ,
constantsVector , constantsValuesVector , inputs2Study );
```

3.2.10 Finding Independent Loops

The function calls another function named *findIndependentCycles* to compute the following matrices and scalar:

- *allCyclesVerticesMatrix*: all loops found in the model in terms of the variables (Vertices) they pass through
- *independentCyclesVerticesMatrix*: the linearly independent loops in terms of the variables (Vertices) they pass through
- *independentCyclesEdgesMatrix*: the linearly independent loops in terms of the links (Edges) they pass through
- *numberIndependentCycles*: number of linearly independent loops in the model

For more information about the internals of this process, the reader should refer to the section of *findIndependentCycles* function.

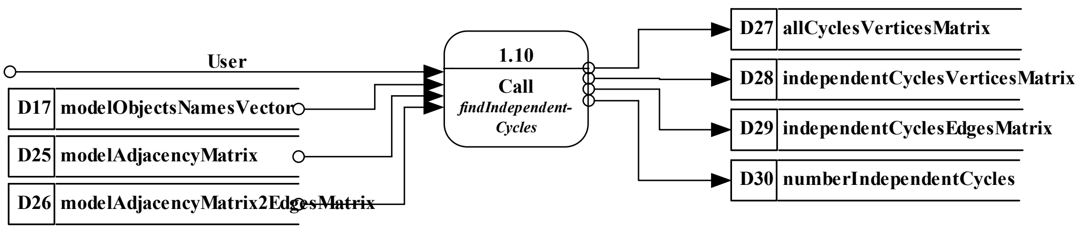


Figure 27: The DFD Level One: Finding Independent Loops

```
% Finding Independent Loops
[ allCyclesVerticesMatrix , ...
  independentCyclesVerticesMatrix , ...
  independentCyclesEdgesMatrix , ...
  numberIndependentCycles ] = findIndependentCycles(
modelAdjacencyMatrix , modelAdjacencyMatrix2EdgesMatrix ,
modelObjectsNamesVector );
```

3.2.11 Applying Eigenvalue Analysis at the Selected Time Steps

The following steps would be repeated for to the model for every user-selected time step using for-loop.

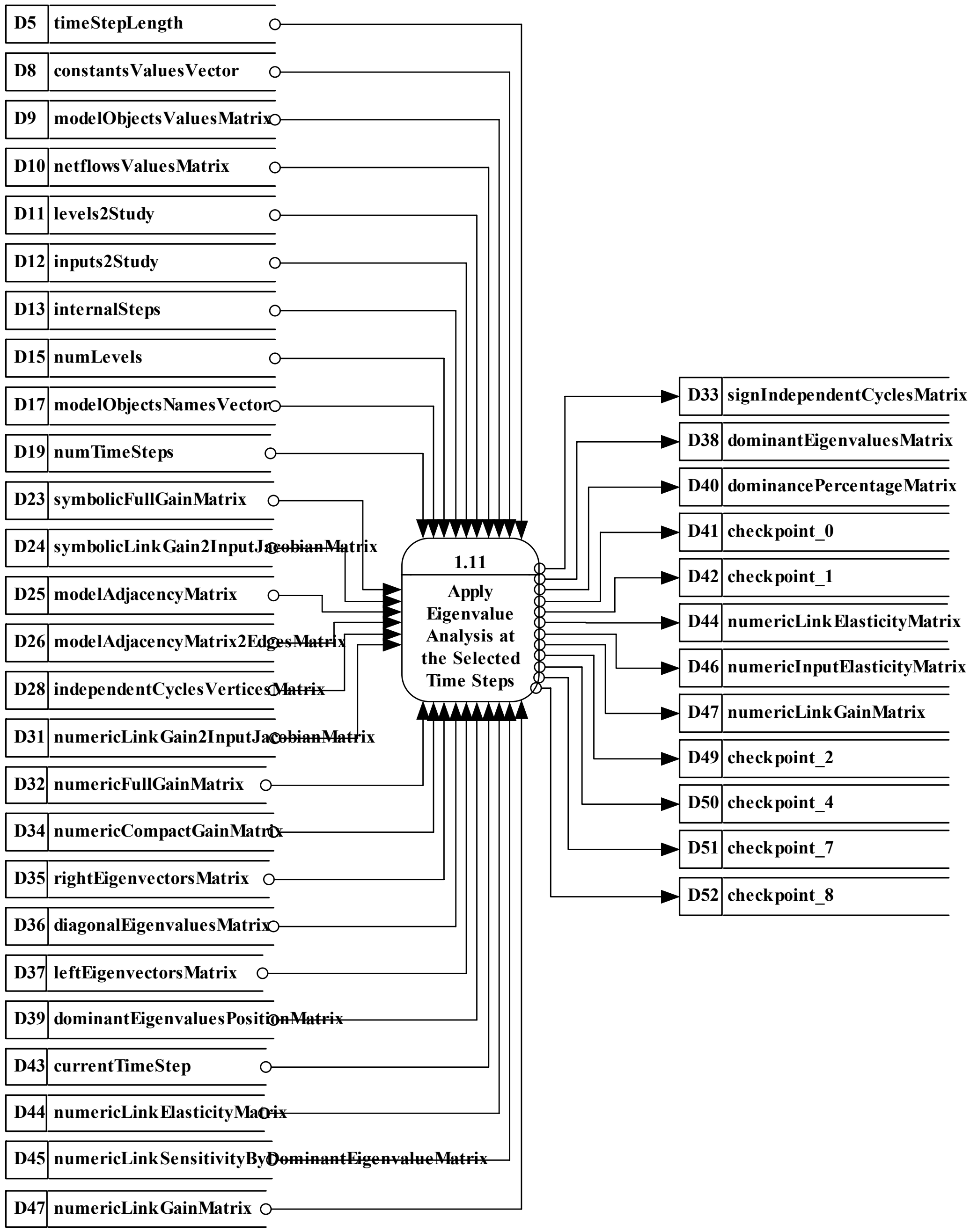


Figure 28: The DFD Level One: Applying Eigenvalue Analysis at the Selected Time Steps

```
% Eigenvalue Analysis at the Selected Time Steps
for currentTimeStep = internalSteps,
    ...
end
```

3.2.11.1 Computing Numeric Full Gain Matrix and Numeric Links' Gains to Inputs Jacobian Matrix

The full gain matrix *symbolicFullGainMatrix* and the links' gains to inputs Jacobian *symbolicLinkGain2InputJacobianMatrix* have been computed in a symbolic form in a previous section, in this section the function substitutes all symbolic variables with their corresponding values to convert the matrices from symbolic into numeric form.

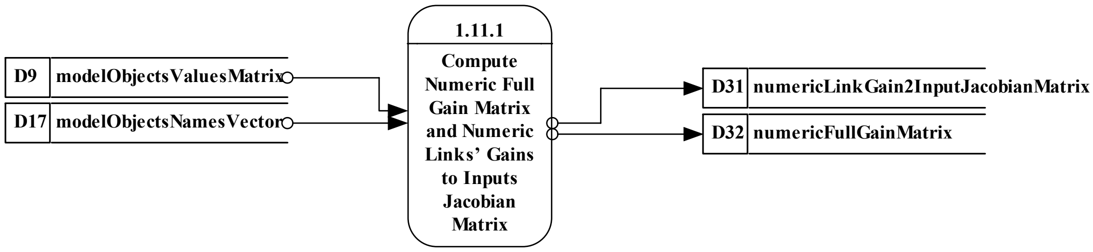


Figure 29: The DFD Level Two: Computing Numeric Full Gain Matrix and Numeric Links' Gains to Inputs Jacobian Matrix

```
% Computing Numeric Full Gain Matrix and Numeric Links' Gains to
Inputs Jacobian Matrix
numericLinkGain2InputJacobianMatrix = double(
subs(symbolicLinkGain2InputJacobianMatrix , modelObjectsNamesVector ,
modelObjectsValuesMatrix( currentTimeStep , : ) ) );
numericFullGainMatrix = double( subs(symbolicFullGainMatrix ,
modelObjectsNamesVector , modelObjectsValuesMatrix( currentTimeStep ,
: ) ) );
```

3.2.11.2 Computing Polarity of the Linearly Independent Loops

The polarity of a loop could be computed by multiplying its links' gains (taking their signs into consideration). The sign of the final result (loop gain) is the polarity of that loop.

For more information about the internals of this process, the reader should refer to the section of *computePathsGain* function.

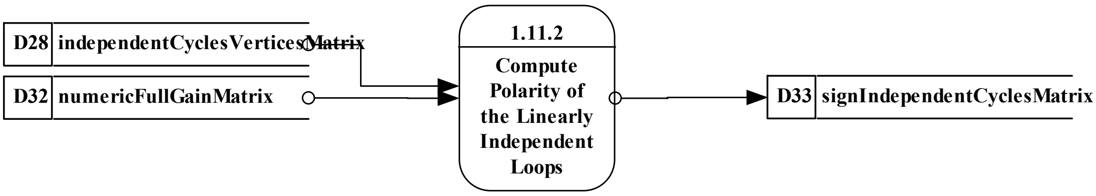


Figure 30: The DFD Level Two: Computing Polarity of the Linearly Independent Loops

```
% Computing Polarity of the Linearly Independent Loops
signIndependentCyclesMatrix( currentTimeStep , : ) = sign(
computePathsGain( numericFullGainMatrix ,
independentCyclesVerticesMatrix ) );
```

3.2.11.3 Computing Compact Gain Matrix

The function divides the full gain matrix into 4 divisions: $\mathbf{A}_{11}$, $\mathbf{A}_{12}$, $\mathbf{A}_{21}$ and $\mathbf{A}_{22}$ according to the following relation:

$$\left[\begin{array}{c|c}\mathbf{A}_{11} & \mathbf{A}_{12} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22}\end{array}\right] = \left.\left[\begin{array}{c|c}\mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} & \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} \\ \hline \mathbf{J}_{\mathbf{z},\mathbf{x}} & \mathbf{J}_{\mathbf{z},\mathbf{z}}\end{array}\right]\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$$

Where: $\left.\left[\begin{array}{c|c}\mathbf{J}_{\dot{\mathbf{x}},\mathbf{x}} & \mathbf{J}_{\dot{\mathbf{x}},\mathbf{z}} \\ \hline \mathbf{J}_{\mathbf{z},\mathbf{x}} & \mathbf{J}_{\mathbf{z},\mathbf{z}}\end{array}\right]\right|_{\tilde{\mathbf{x}},\tilde{\mathbf{z}},\tilde{\mathbf{u}}}$ is the full gain matrix (System Jacobian).

Then, the function computes the compact gain matrix in a numeric form *numericCompactGainMatrix* (the gain matrix of the compact version of the model) according to the following equation:

$$numericCompactGainMatrix = \mathbf{A}_{11} + \mathbf{A}_{12}(\mathbf{I} - \mathbf{A}_{22})^{-1}\mathbf{A}_{21}$$

But the relation: $\mathbf{A}_{11} = \mathbf{0}$, is always valid:

$$\therefore numericCompactGainMatrix = \mathbf{A}_{12}(\mathbf{I} - \mathbf{A}_{22})^{-1}\mathbf{A}_{21}$$

For more information the reader should refer to the mathematical background chapter.

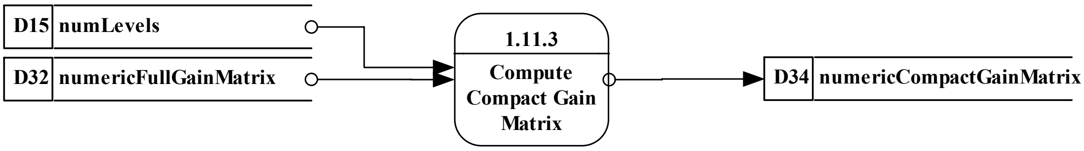


Figure 31: The DFD Level Two: Computing Compact Gain Matrix

```
% Computing Compact Gain Matrix
% [ A11           A12 ]
% [                   ]
% [ A21           A22 ]
%
%  m *            n *
%     m              m
%
%  m *            n *
%     n              n
%
% m  = length( levelsVector )
% n  = length( auxiliariesVector )
% Note: A11 will always be a null matrix
A12 = numericFullGainMatrix( 1 : numLevels , numLevels+1 : end );
A21 = numericFullGainMatrix( numLevels+1 : end , 1 : numLevels );
A22 = numericFullGainMatrix( numLevels+1 : end , numLevels + 1 : end
);
numericCompactGainMatrix = A12 * inv( eye( size( A22 ) ) - A22 ) *
A21;
```

3.2.11.4 Computing Eigenvalues and Eigenvectors of the Compact Gain Matrix

After computing the compact gain matrix in a symbolic form, it would be easy for function to compute its eigenvalues and right eigenvector (using Matlab internal function *eig*), and then to compute its left eigenvector by computing the transpose of the inverse of its right eigenvector.

$$\mathbf{l} = \left(\mathbf{r}^{-1}\right)^T$$

Where: $\mathbf{l}$ and $\mathbf{r}$ are the left and right eigenvectors respectively.

For more information the reader should refer to the mathematical background chapter.

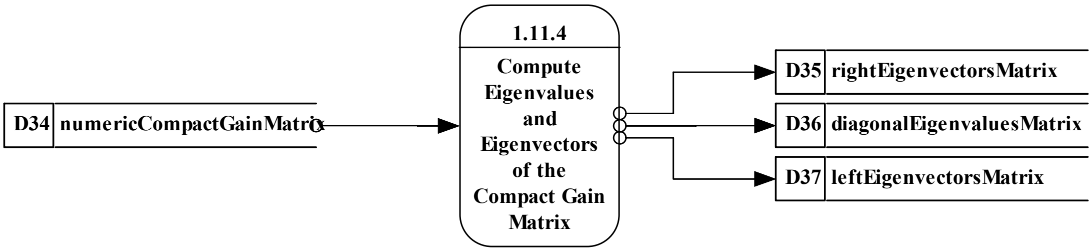


Figure 32: The DFD Level Two: Computing Eigenvalues and Eigenvectors of the Compact Gain Matrix

```
% Computing Eigenvalues and Eigenvectors of the Compact Gain Matrix
tempNumericCompactGainMatrix = sym( numericCompactGainMatrix , 'd' );
[ rightEigenvectorsMatrix , ...
  diagonalEigenvaluesMatrix ] = eig( tempNumericCompactGainMatrix );
rightEigenvectorsMatrix = double( rightEigenvectorsMatrix );
diagonalEigenvaluesMatrix = double( diagonalEigenvaluesMatrix );
leftEigenvectorsMatrix = inv( rightEigenvectorsMatrix ).';
```

3.2.11.5 Identifying Dominant Eigenvalue

The function calls another function named *findDominantEigenvalue* to compute the following matrices and vectors:

- *dominantEigenvaluesMatrix*: the matrix of the values of eigenvalues for every time step sorted according to their dominance
- *dominantEigenvaluesPositionMatrix*: the matrix of positions of eigenvalues for every time step sorted according to their dominance
- *dominancePercentageMatrix*: the matrix of dominance percentage of eigenvalues for every time step sorted according to their dominance
- *tempCheckpoint_0*: a vector to carry the data of checkpoint 0
- *tempCheckpoint_1*: a vector to carry the data of checkpoint 1

Then the function uses the variables *tempCheckpoint_0* and *tempCheckpoint_1* to fill *checkpoint_0* and *checkpoint_1* respectively.

For more information about the internals of this process, the reader should refer to the section of *findDominantEigenvalue* function.

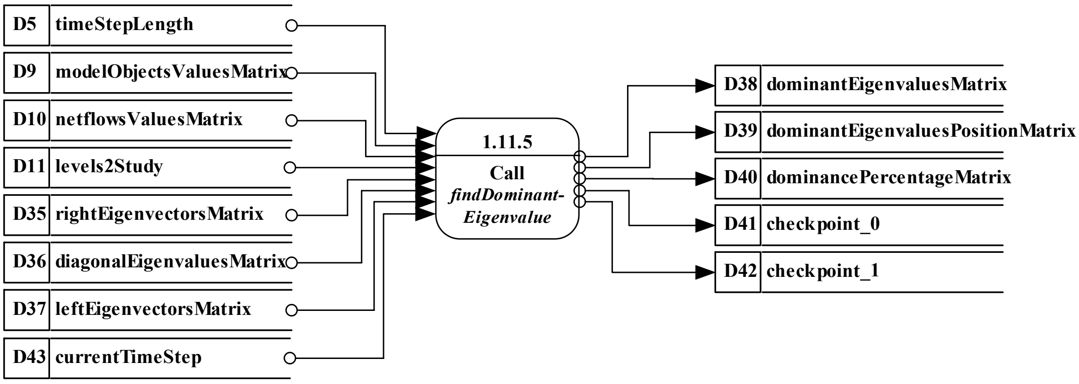


Figure 33: The DFD Level Two: Identifying Dominant Eigenvalue

```
% Identifying Dominant Eigenvalue
[ dominantEigenvaluesMatrix( currentTimeStep , : ) , ...
  dominantEigenvaluesPositionMatrix( currentTimeStep , : ) , ...
  dominancePercentageMatrix( currentTimeStep , : ) , ...
  tempCheckpoint_0 , ...
  tempCheckpoint_1 ] = findDominantEigenvalue(
rightEigenvectorsMatrix , leftEigenvectorsMatrix ,
diagonalEigenvaluesMatrix , netflowsValuesMatrix( currentTimeStep , :
).' , netflowsValuesMatrix( currentTimeStep + 1 , : ).' ,
modelObjectsValuesMatrix( currentTimeStep , 1 : numLevels ).' ,
modelObjectsValuesMatrix( currentTimeStep + 1 , 1 : numLevels ).' ,
timeStepLength , levels2Study , currentTimeStep );
checkpoint_0 = [ checkpoint_0 ; tempCheckpoint_0(:).' ];
checkpoint_1 = [ checkpoint_1 ; tempCheckpoint_1(:).' ];
```

3.2.11.6 Computing Links' Elasticity Values associated with the Dominant Eigenvalue

The function calls another function named *computeLinkElasticity* to compute the following matrices and vectors:

- *numericLinkElasticityMatrix*: a matrix of links' elasticity values associated with the dominant eigenvalue

- *numericLinkSensitivityByDominantEigenvalueMatrix*: a matrix of links' sensitivity values associated with the dominant eigenvalue divided by the value of the dominant eigenvalue
- *tempCheckpoint_2*: a vector to carry the data of checkpoint 2
- *tempCheckpoint_4*: a vector to carry the data of checkpoint 4
- *tempCheckpoint_7*: a vector to carry the data of checkpoint 7
- *tempCheckpoint_8*: a vector to carry the data of checkpoint 8

Then the function uses the variables *tempCheckpoint_2*, *tempCheckpoint_4*, *tempCheckpoint_7* and *tempCheckpoint_8* to fill *checkpoint_2*, *checkpoint_4*, *checkpoint_7* and *checkpoint_8* respectively.

For more information about the internals of this process, the reader should refer to the section of *computeLinkElasticity* function.

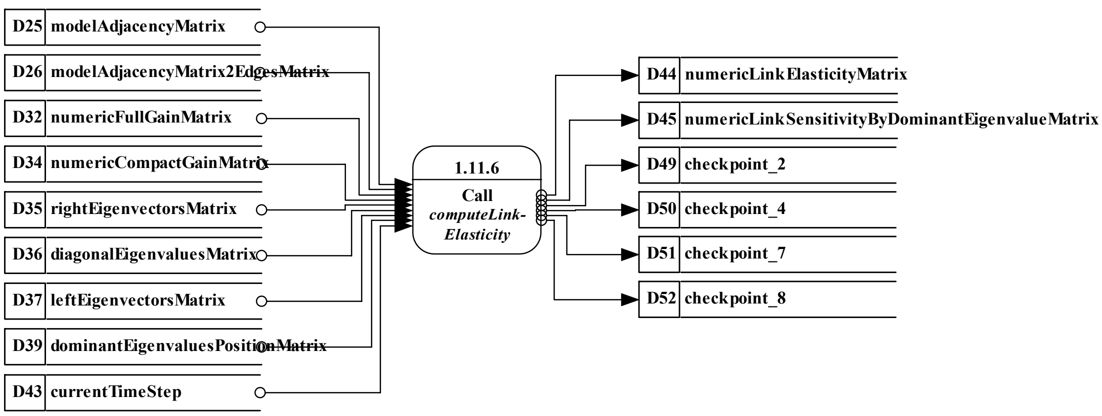


Figure 34: The DFD Level Two: Computing Links' Elasticity Values associated with the Dominant Eigenvalue

```
% Computing Links' Elasticity Values associated with the Dominant
Eigenvalue
[ numericLinkElasticityMatrix( : , currentTimeStep ) , ...
  numericLinkSensitivityByDominantEigenvalueMatrix( : ,
currentTimeStep ) , ...
  tempCheckpoint_2 , ...
  tempCheckpoint_4  , ...
  tempCheckpoint_7 , ...
```

```
  tempCheckpoint_8 ] = computeLinkElasticity(
numericCompactGainMatrix , numericFullGainMatrix ,
modelAdjacencyMatrix , modelAdjacencyMatrix2EdgesMatrix ,
rightEigenvectorsMatrix , leftEigenvectorsMatrix ,
diagonalEigenvaluesMatrix , dominantEigenvaluesPositionMatrix(
currentTimeStep , 1 ) , currentTimeStep );
checkpoint_2 = [ checkpoint_2 ; tempCheckpoint_2(:).' ];
checkpoint_4 = [ checkpoint_4 ; tempCheckpoint_4(:).' ];
checkpoint_7 = [ checkpoint_7 ; tempCheckpoint_7(:).' ];
checkpoint_8 = [ checkpoint_8 ; tempCheckpoint_8(:).' ];
```

3.2.11.7 Computing Inputs' Elasticity Values associated with the Dominant Eigenvalue

The function calls another function named *computeInputElasticity* to compute the following matrix:

- *numericInputElasticityMatrix*: a matrix of inputs' elasticity values associated with the dominant eigenvalue

For more information about the internals of this process, the reader should refer to the section of *computeInputElasticity* function.

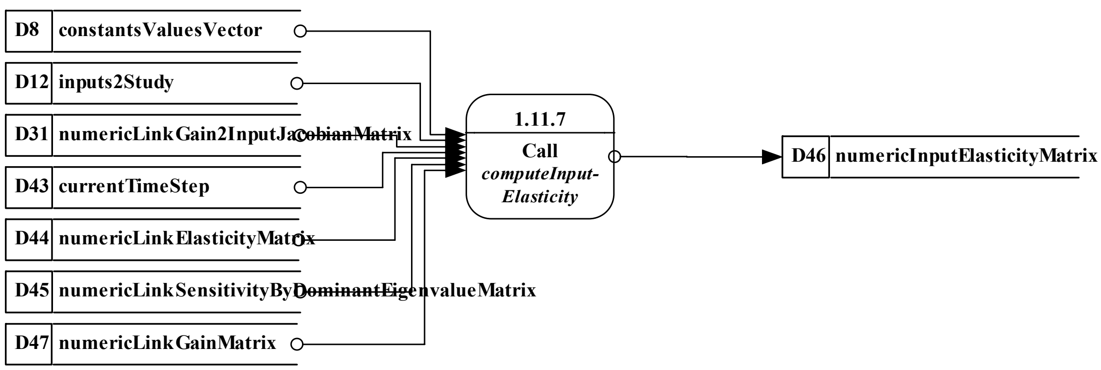


Figure 35: The DFD Level Two: Computing Inputs' Elasticity Values associated with the Dominant Eigenvalue

```
% Computing Inputs' Elasticity Values associated with the Dominant
Eigenvalue
numericInputElasticityMatrix( : , currentTimeStep ) =
computeInputElasticity( numericLinkGainMatrix( : , currentTimeStep )
, numericLinkGain2InputJacobianMatrix , numericLinkElasticityMatrix(
: , currentTimeStep ) , constantsValuesVector , inputs2Study ,
```

```
numericLinkSensitivityByDominantEigenvalueMatrix( : , currentTimeStep
) );
```

3.2.12 Computing Linearly Independent Loops' Elasticity Values associated with the Dominant Eigenvalue

The function calls another function named *computeIndependentCycleElasticity* to compute the following matrix:

- *independentCyclesElasticityMatrix*: a matrix of linearly independent loops' elasticity values associated with the dominant eigenvalue

For more information about the internals of this process, the reader should refer to the section of *computeIndependentCycleElasticity* function.

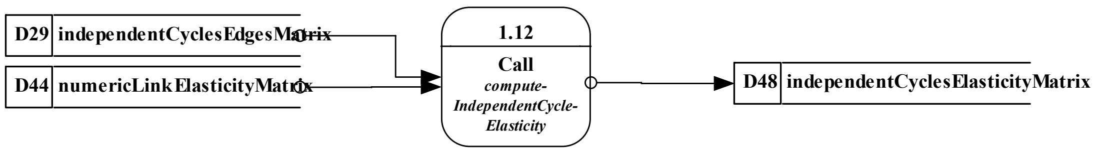


Figure 36: The DFD Level One: Computing Independent Loops' Elasticity Values associated with the Dominant Eigenvalue

```
% Computing Independent Loops' Elasticity Values associated with the
Dominant Eigenvalue
independentCyclesElasticityMatrix =
computeIndependentCycleElasticity( independentCyclesEdgesMatrix ,
numericLinkElasticityMatrix );
```

3.2.13 Ending and Closing Checkpoints

The following code listing contains the process of ending and closing of only one checkpoint file. The other checkpoint files have the same lines of code to perform file ending and closing.

The function performs the following steps:

- Computing mean, mean absolute, maximum and minimum of checkpoint values over the time range

- Opening the checkpoint file in the append mode; to write to the file without overwriting it
- Writing the needed lines of the previously computed values
- Closing the file

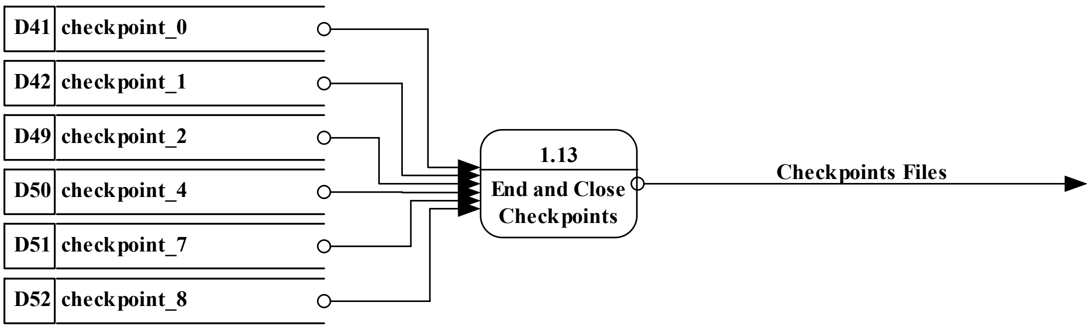


Figure 37: The DFD Level One: Ending and Closing Checkpoints

```
% Ending and Closing Checkpoint(1)
mean_checkpoint_1 = mean( checkpoint_1 );
mean_abs_checkpoint_1 = mean( abs( checkpoint_1 ) );
max_checkpoint_1 = max( checkpoint_1 );
min_checkpoint_1 = min( checkpoint_1 );
fid = fopen( [ 'checkpoint_1.csv' ] , 'a' );
fwrite( fid , [ sprintf( '\n' ) 'Mean;' ] );
for I = 1 : 2 * numLevels,
    fwrite( fid , [ num2str( mean_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf( '\n' ) 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
    fwrite( fid , [ num2str( mean_abs_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf( '\n' ) 'Max;' ] );
for I = 1 : 2 * numLevels,
    fwrite( fid , [ num2str( max_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf( '\n' ) 'Min;' ] );
for I = 1 : 2 * numLevels,
    fwrite( fid , [ num2str( min_checkpoint_1( I ) ) ';' ] );
end
fclose( fid );
```

3.2.14 Printing to Output File

The function calls another function named *printOutputs* to print all its outputs into the output file using the output file name specified by the user.

For more information about the internals of this process, the reader should refer to the section of *printOutputs* function.

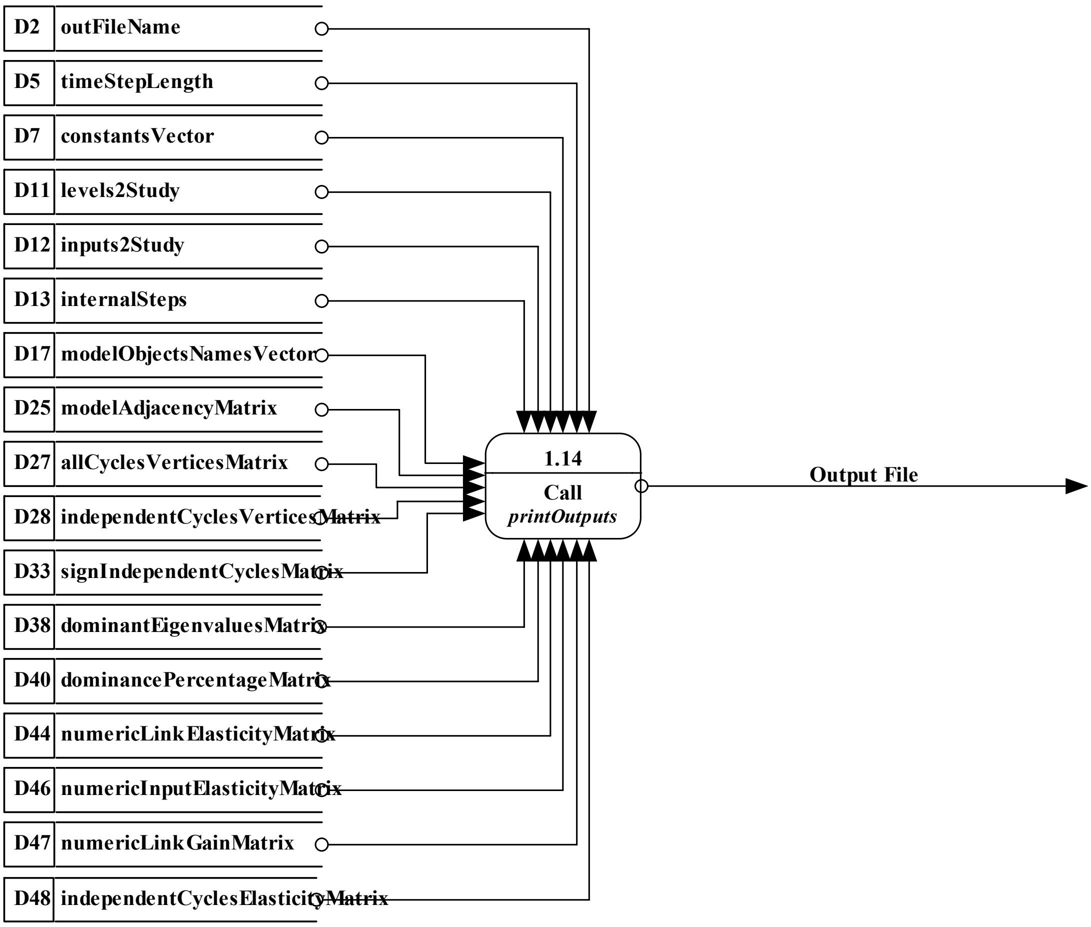


Figure 38: The DFD Level One: Printing to Output File

```
% Printing to Output File
printOutputs( levels2Study , inputs2Study , modelAdjacencyMatrix ,
internalSteps , timeStepLength , dominantEigenvaluesMatrix ,
dominancePercentageMatrix , numericLinkGainMatrix ,
numericLinkElasticityMatrix , numericInputElasticityMatrix ,
independentCyclesElasticityMatrix , allCyclesVerticesMatrix ,
independentCyclesVerticesMatrix , signIndependentCyclesMatrix ,
modelObjectsNamesVector , constantsVector , outFileName );
```

3.3 The extractModelObjects Function

The *extractModelObjects* function extracts information from inputs comes from the Simulation package to the Analysis package; the vector of structures *modelObjectsStructVector*.

An element of the vector *modelObjectsStructVector* has four fields:

- *Name*: the name of the object the structure contains
- *Equation*: the equation of that object
- *Value*: a variable used to contain an instantaneous values for calculations of that object in previous step (inside the Simulation package)
- *State*: a Boolean variable that equals one if the object is a state and zero for any other object

The function extracts its outputs using that information as would be explained in the next sections.

3.3.1 Computing Number of Levels and Auxiliaries

The number of levels *numLevels* is computed by summing the field *state* of all elements of the *modelObjectsStructVector*; i.e. *modelObjectsStructVector.state*. While the number of auxiliaries *numAuxiliaries* is computed by summing the logical invert of the field *state* of all elements of the *modelObjectsStructVector*.

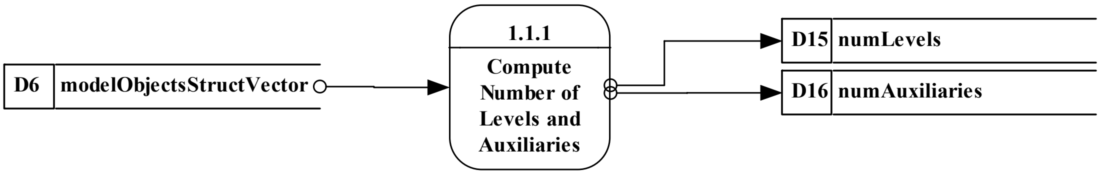


Figure 39: The DFD Level Three: Computing Number of Levels and Auxiliaries

```
% Computing Number of Levels and Auxiliaries
numLevels = sum( [ modelObjectsStructVector.state ] );
numAuxiliaries = sum( ~[ modelObjectsStructVector.state ] );
```

3.3.2 Extracting Objects' Names and Equations

The vector of objects' names *modelObjectsNamesVector* is computed by collecting the field *name* of all elements of the *modelObjectsStructVector* in a vector; i.e. collecting all *modelObjectsStructVector.name*. While the vector of objects' equations *modelObjectsEquationsVector* is computed by collecting the field *equation* of all elements of the *modelObjectsStructVector* in a vector; i.e. collecting all *modelObjectsStructVector.equation*.

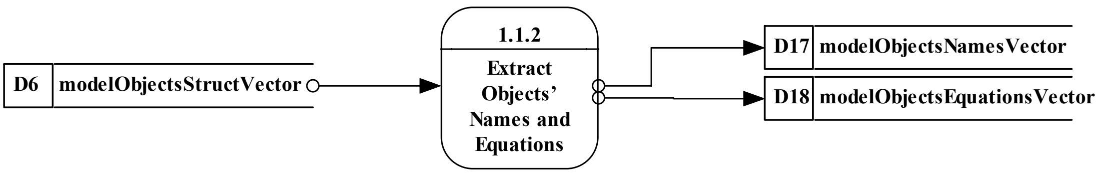


Figure 40: The DFD Level Three: Extracting Objects' Names and Equations

```
% Extracting Objects' Names and Equations
modelObjectsNamesVector = [ modelObjectsStructVector.name ];
modelObjectsEquationsVector = [ modelObjectsStructVector.equation ];
```

3.4 The computeSystemJacobians Function

The *computeSystemJacobians* function computes two Jacobian matrices in symbolic form; one of them is the full gain matrix *symbolicFullGainMatrix*, the other one is the Jacobian of the vector of the links' gains to the vector of the inputs *symbolicLinkGain2InputJacobianMatrix*, this Jacobian matrix is used in the calculations of the inputs' elasticity values associated with dominant eigenvalue.

Moreover the function computes two other important matrices; the model adjacency matrix *modelAdjacencyMatrix* and the model adjacency matrix to edges matrix *modelAdjacencyMatrix2EdgesMatrix*, which is a dictionary matrix used to find the link number if its start and end variables are known and vice versa.

3.4.1 Computing Symbolic Full Gain Matrix

The function calls another function named *jac* to compute the full gain matrix in symbolic form, which is the Jacobian of the vector of equations of net-flows and auxiliaries concatenated to the vector of names of states and auxiliaries concatenated, and before it ends; it substitutes the constants with their corresponding values to compute the final result of the symbolic full gain matrix.

The function performs this calculation according to the following equation:

$$symbolicFullGainMatrix = \left[\begin{array}{c|c} \frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{x}} & \frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{z}} \\ \hline \frac{\partial \mathbf{z}}{\partial \mathbf{x}} & \frac{\partial \mathbf{z}}{\partial \mathbf{z}} \end{array}\right]$$

Where:

$\dot{\mathbf{x}}$: is the vector of net-flows

$\mathbf{x}$: is the vector of states

$\mathbf{z}$: is the vector of auxiliaries

The last equation could be simplified to:

$$symbolicFullGainMatrix = \left[\frac{\partial \left[\dot{\mathbf{x}}^T \;\vdots\; \mathbf{z}^T \right]}{\partial \left[\mathbf{x}^T \;\vdots\; \mathbf{z}^T \right]} \right]$$

But: $modelObjectsEquationsVector = \left[\dot{\mathbf{x}}^T \;\vdots\; \mathbf{z}^T \right]$

and $modelObjectsNamesVector = \left[\mathbf{x}^T \;\vdots\; \mathbf{z}^T \right]$.

$$\therefore symbolicFullGainMatrix = \left[\frac{\partial \left(modelObjectsEquationsVector \right)}{\partial \left(modelObjectsNamesVector \right)} \right]$$

For more information about the internals of this process, the reader should refer to the section of *jac* function.

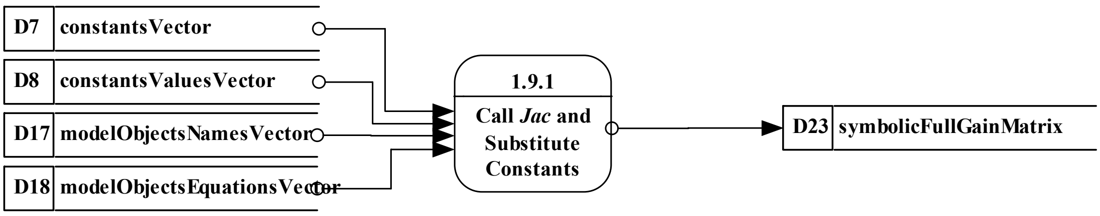


Figure 41: The DFD Level Three: Computing Symbolic Full Gain Matrix

```
% Computing Symbolic Full Gain Matrix
symbolicFullGainMatrix = jac( modelObjectsEquationsVector.' ,
modelObjectsNamesVector );
.
.
.
symbolicFullGainMatrix = subs( symbolicFullGainMatrix ,
constantsVector , constantsValuesVector );
```

3.4.2 Computing Model Adjacency Matrix

The function creates the model adjacency matrix by creating a matrix of zeros that has the same size of the full gain matrix and has the same zero elements, while replacing the non-zero elements by ones.

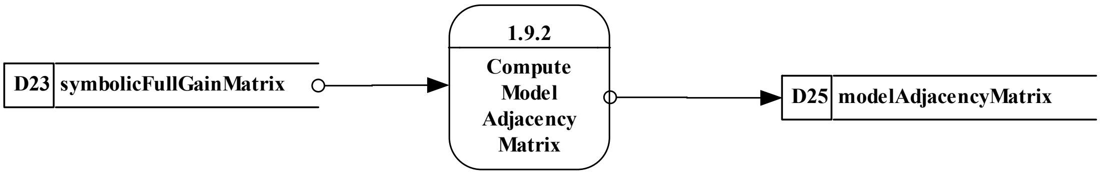


Figure 42: The DFD Level Three: Computing Model Adjacency Matrix

```
% Computing Model Adjacency Matrix
modelAdjacencyMatrix = zeros( size( symbolicFullGainMatrix ) );
modelAdjacencyMatrix( find( symbolicFullGainMatrix ~= 0 ) ) = 1;
```

3.4.3 Computing Model Adjacency Matrix to Edges Matrix

The *computeSystemJacobians* function creates the model adjacency matrix to edges matrix by replacing each one in the model adjacency matrix by its number among the other ones.

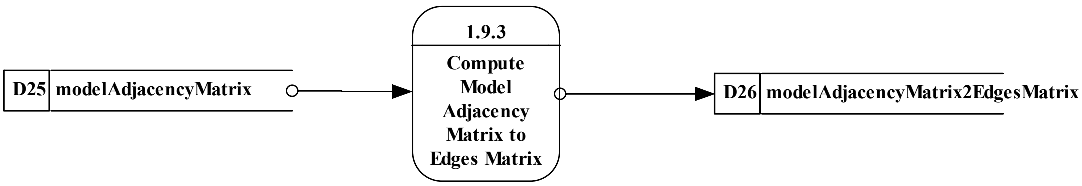


Figure 43: The DFD Level Three: Computing Model Adjacency Matrix to Edges Matrix

```
% Computing Model Adjacency Matrix to Edges Matrix
modelAdjacencyMatrix2EdgesMatrix = modelAdjacencyMatrix;
modelAdjacencyMatrix2EdgesMatrix( find( modelAdjacencyMatrix ~= 0 ) )
= [ 1 : nnz( modelAdjacencyMatrix ) ];
```

3.4.4 Computing Symbolic Link Gain to Input Jacobian Matrix

The *computeSystemJacobians* function collects the links' gain in symbolic form from the symbolic full gain matrix and puts them into a vector, then it uses *jac* function to compute the link gain to input Jacobian matrix in symbolic form, and then replaces the constants with their corresponding values.

For more information about *jac* function the reader should refer its section.

$$symbolicLinkGain2InputJacobianMatrix = \left[\frac{\partial \ell}{\partial \mathbf{u}}\right]$$

Where: ℓ vector of links and $\mathbf{u}$ vector of inputs.

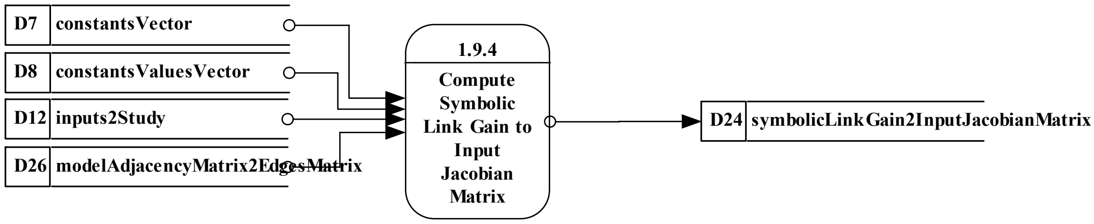


Figure 44: The DFD Level Three: Computing Symbolic Link Gain to Input Jacobian Matrix

```
% Computing Symbolic Link Gain to Input Jacobian Matrix
[ x , y ] = find( modelAdjacencyMatrix2EdgesMatrix ~= 0 );
for I = 1 : length( x ),
    symbolicLinkGainVector( modelAdjacencyMatrix2EdgesMatrix( x( I )
, y( I ) ) ) = symbolicFullGainMatrix( x( I ) , y( I ) );
end
symbolicLinkGain2InputJacobianMatrix = jac( symbolicLinkGainVector ,
constantsVector( inputs2Study ) );
symbolicLinkGain2InputJacobianMatrix = subs(
symbolicLinkGain2InputJacobianMatrix , constantsVector ,
constantsValuesVector );
```

3.5 The findIndependentCycles Function

The function finds all loops in the model, also it finds a set of linearly independent loops (Kampmann, C. E., 1996), by letting the user to choose a set of important loops from his/her point of view out of the all loops set, then the function tries to construct a set of linearly independent loops that contains the user-selected loops as much as possible and completes this set with the shortest possible loops.

3.5.1 Finding All Loops

The *findIndependentCycles* function uses *allcycsn* function to find all loops in the model. For more information the reader should refer to the "Digraph toolbox" (Bahar, M.; Jantzen, J., 1995).

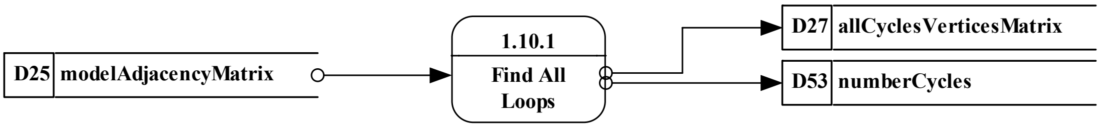


Figure 45: The DFD Level Three: Finding All Loops

```
% Finding All Loops
allCyclesVerticesMatrix = allcycsn( modelAdjacencyMatrix );
numberCycles = size( allCyclesVerticesMatrix , 1 );
```

3.5.2 Computing the Cycles' Matrix

The cycles' matrix is the matrix where all cycles (loops) are expressed in binary form by the links they pass through and not by the variables they pass through (Kampmann, C. E., 1996).

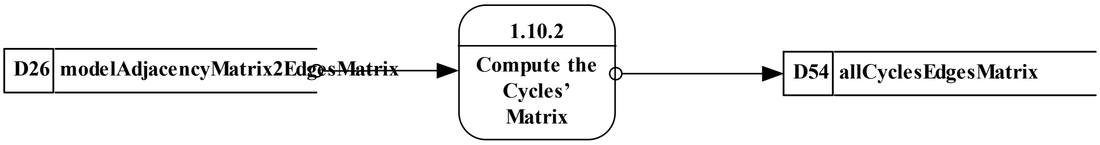


Figure 46: The DFD Level Three: Computing the Cycles' Matrix

```
% Computing the Cycles' Matrix
allCyclesEdgesMatrix = zeros( numberCycles , max( max (
modelAdjacencyMatrix2EdgesMatrix ) ) );
for I = 1 : numberCycles,
    oneCycle = nonzeros( allCyclesVerticesMatrix( I , : ) ).';
    for J = 1 : size( oneCycle , 2 ) - 1,
        K = modelAdjacencyMatrix2EdgesMatrix( oneCycle( J + 1 ) ,
oneCycle( J ) );
        allCyclesEdgesMatrix( I , K ) = 1;
    end
end
```

3.5.3 User-interaction: Suggesting Loops to be tested for Linear Independency

Using a code similar to that of the user-interaction to choose inputs and level to study in previous section, the *findIndependentCycles* function prints all loops in the model to the user, and let him/her to choose a set of important loops from his/her point of view in a vector form.

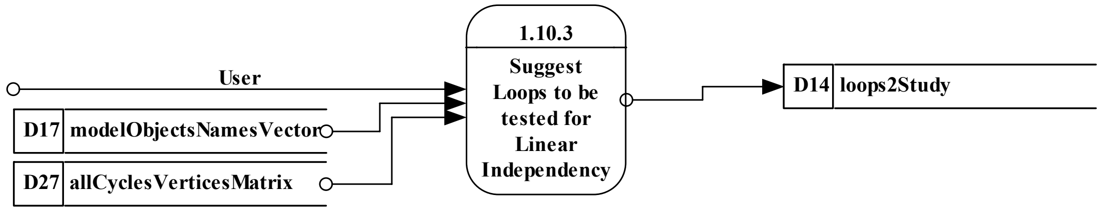


Figure 47: The DFD Level Three: Suggesting Loops to be tested for Linear Independency

```
% User-interaction: Suggesting Loops to be tested for Linear
Independency
endLoop = true;
while ( endLoop ),
    % Printing All Loops
    disp( [ sprintf( '\n' ) 'All Loops:' sprintf( '\n' ) ] );
    for I = 1:size( allCyclesVerticesMatrix , 1 ) ,
        tempPrint = [];
        oneIndependentCycle = nonzeros( allCyclesVerticesMatrix( I ,
: ) ).';
        disp( [ 'Loop' sprintf( '\t' ) num2str( I ) ':' sprintf( '\t'
) ] );
        for J = 1:size( oneIndependentCycle , 2 )-1 ,
            tempPrint = [ tempPrint , char( modelObjectsNamesVector(
oneIndependentCycle( J ) ) ) ];
            if J ~= size( oneIndependentCycle , 2 )-1 ,
                tempPrint = [ tempPrint , ' --> ' ];
            end
        end
        disp( tempPrint );
    end
    loops2Study = input( [ 'Enter the number(s) of the Loop(s) you
are intersted' sprintf( '\n' ) 'in studying in a vector form  (ex.:
[1,2,6]):' sprintf( '\t' ) ] );
    if max( loops2Study ) > size( allCyclesVerticesMatrix , 1 ) |
min( loops2Study ) < 1 | size( loops2Study , 1 ) ~= 1,
        disp( 'Wrong Input(s), try again ...' );
    else
        endLoop = false;
    end
end
```

3.5.4 Identifying Independent Cycles

The *findIndependentCycles* function computes the number of the linearly independent loops set by computing the binary rank of the all loops matrix (Kampmann, C. E., 1996).

```
numberIndependentCycles = rank( allCyclesEdgesMatrix );
```

Then the function puts the user-selected loops in the previous step to the top of the loops matrix and after these loops the rest of the loops come in their original order which is an ascending order according to the length of the loops. Then it tries to construct a set of linearly independent loops by removing a loop from the matrix and test its rank, which has two cases:

1. The rank remains the same; which means that the removed loop is linearly dependant on the other remaining loops, and goes for testing another loop
2. The rank changes; which means that the removed loop is linearly independent on the other remaining loops, and then the functions retrieves that loop and goes for testing another loop

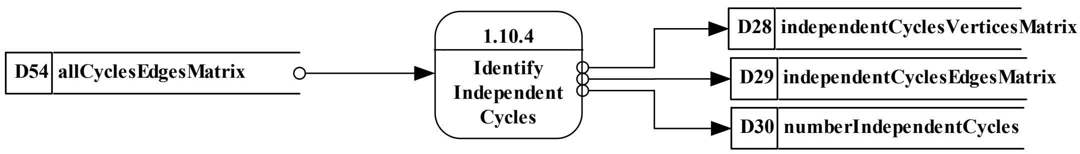


Figure 48: The DFD Level Three: Identifying Independent Cycles

```
independentCyclesEdgesMatrix = allCyclesEdgesMatrix;
independentCyclesVerticesMatrix = allCyclesVerticesMatrix;
temp1 = independentCyclesEdgesMatrix( loops2Study , : );
temp2 = independentCyclesVerticesMatrix( loops2Study , : );
independentCyclesEdgesMatrix( loops2Study , : ) = [];
independentCyclesVerticesMatrix( loops2Study , : ) = [];
independentCyclesEdgesMatrix = [ temp1 ; independentCyclesEdgesMatrix
];
```

```
independentCyclesVerticesMatrix = [ temp2 ;
independentCyclesVerticesMatrix ];
...
...
...
for I = 1 : numberCycles,
    tempCycles = independentCyclesEdgesMatrix;
    tempCyclesn = independentCyclesVerticesMatrix;
    independentCyclesEdgesMatrix( I , : ) = 0;
    independentCyclesVerticesMatrix( I , : ) = 0;
    if ~( rank( independentCyclesEdgesMatrix ) ==
numberIndependentCycles ),
        independentCyclesEdgesMatrix = tempCycles;
        independentCyclesVerticesMatrix = tempCyclesn;
    end
end
```

3.6 The findDominantEigenvalue Function

The function aims at identifying the eigenvalue that dominates the behavior of the selected level to study.

3.6.1 Computing Analysis Time Step Length

The length of the analysis time step is taken to be equal to the simulation time step length.

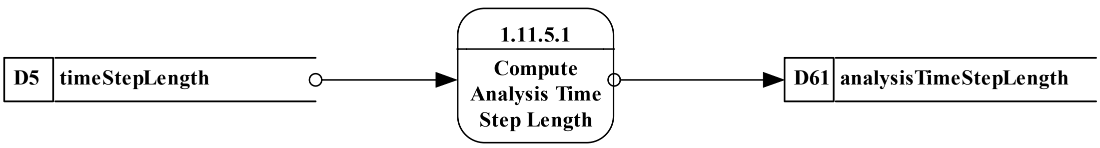


Figure 49: The DFD Level Four: Computing Analysis Time Step Length

```
% Analysis Time Step Length
analysisTimeStepLength = timeStepLength;
```

3.6.2 Computing Contributions of Eigenvalues

As stated in the mathematical background chapter, there are two cases that could result when computing the change in the value of a level due to each eigenvalue, according to the nature of the eigenvalue:

1. For non-zero eigenvalue; $\lambda_i \neq 0$:

$$\delta x_{ki} = \frac{c_{ik}}{\lambda_i}\left(e^{\lambda_i \delta t} - 1\right)$$

2. For zero eigenvalue; $\lambda_i = 0$:

$$\delta x_{ki} = c_{ik}\,\delta t$$

Where:

δx_{ki}: The change in the kth state due to the ith eigenvalue = *deltaStateTerms*(*i,k*)

c_{ik}: The integration constant in the equation of the kth level due to the ith eigenvalue, called alpha by Saleh, M. and Davidsen, P. (Saleh, M.; Davidsen, P. I., 2000), note that: $c_i = \mathbf{l}_i^T \tilde{\dot{\mathbf{x}}}$ where: $\mathbf{l}_i$ and $\tilde{\dot{\mathbf{x}}}$ are the left eigenvectors associated with the ith eigenvalue and the initial value of the net-flows values vector respectively.

δt: The length of the analysis time step

To compute the ith eigenvalue contribution in the behavior of the kth state contribution_{ki}; the following equation is used:

$$\text{contribution}_{ki} = \frac{\delta x_{ki}}{\delta x_k}$$

These contribution values are stored in one vector, and then sorted in descending order; to get the most dominant eigenvalue.

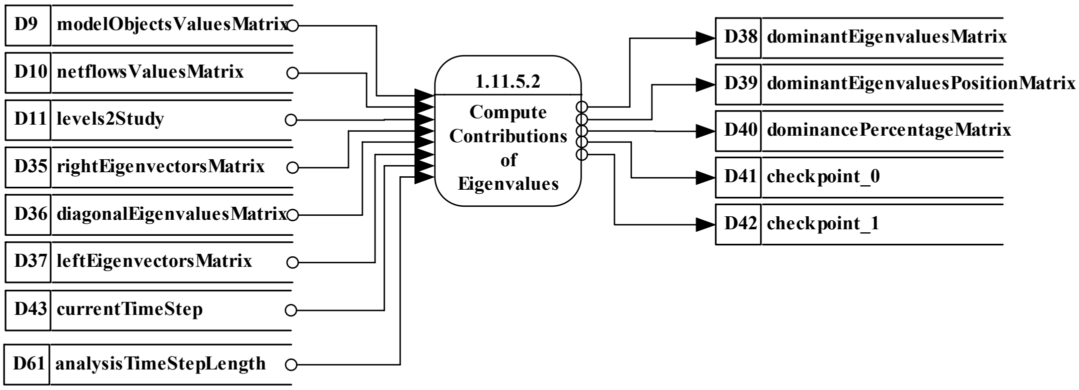


Figure 50: The DFD Level Four: Computing Contributions of Eigenvalues

```
% Computing Contributions of Eigenvalues
alphasVector = leftEigenvectorsMatrix.' * netflowsValuesVector;
deltaStateTerms = ( zeros( size( rightEigenvectorsMatrix ) ) );
...
for K = 1 : numLevels ,
    if eigenvaluesVector( K ) == 0,
        deltaStateTerms( : , K ) = rightEigenvectorsMatrix( : , K )
.* ( alphasVector( K ) * analysisTimeStepLength );
    else
        deltaStateTerms( : , K ) = rightEigenvectorsMatrix( : , K )
.* ( alphasVector( K ) * ( exp( eigenvaluesVector( K ) *
analysisTimeStepLength ) - 1 ) / eigenvaluesVector( K ) );
    end
end
...
deltaState = sum( deltaStateTerms , 2 );
...
for K = 1 : numLevels ,
    flags = zeros( numLevels , 1 );
    flags( K ) = 1;
    if ~isreal( eigenvaluesVector( K ) ),
        conjK = find( eigenvaluesVector == conj( eigenvaluesVector( K
) ) );
        flags( conjK ) = 1;
    end
    deltaStateTerm = sum( deltaStateTerms( levels2Study , : ) .*
flags.' );
    contribution = deltaStateTerm / deltaState( levels2Study );
    dominancePercentageVector = [ dominancePercentageVector , 100 *
real( contribution ) ];
end
[ dominancePercentageVector , dominantEigenvaluesPositionVector ] =
sort( dominancePercentageVector );
```

3.7 The computeLinkElasticity Function

The function computes links' elasticity values associated with dominant eigenvalue.

3.7.1 Computing Sensitivity associated with Dominant Eigenvalue Values

The sensitivity values matrix is computed by the following relation:

$$\mathbf{S}_k = \mathbf{l}_k \mathbf{r}_k^T$$

Where: $\mathbf{l}_k$ and $\mathbf{r}_k$ are the left and right eigenvectors of the kth eigenvalue respectively.

The elasticity values matrix is computed by the following relation:

$$\mathbf{E}_k = \frac{1}{\lambda_k} \mathbf{S}_k \mathbf{.*A}$$

Where: $\mathbf{E}_k$ is the kth eigenvalue elasticity matrix for the system compact gain matrix.

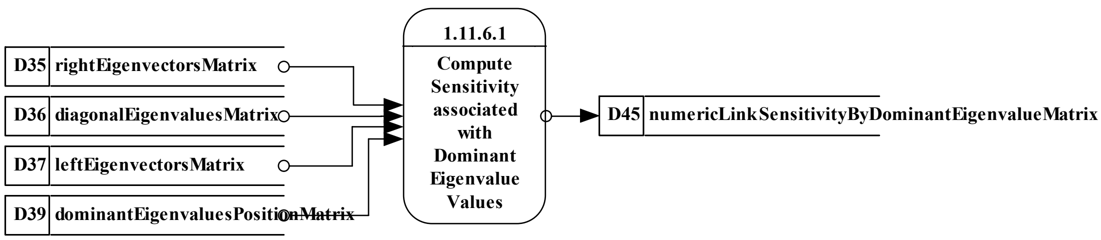


Figure 51: The DFD Level Four: Computing Sensitivity associated with Dominant Eigenvalue Values

```
% Computing Sensitivity associated with Dominant Eigenvalue Values
numericSensitivityMatrix = leftEigenvectorsMatrix( : ,
dominantEigenvaluePosition ) * rightEigenvectorsMatrix( : ,
dominantEigenvaluePosition ).';
numericLinkSensitivityByDominantEigenvalueMatrix =
numericSensitivityMatrix / diagonalEigenvaluesMatrix(
```

```
dominantEigenvaluePosition , dominantEigenvaluePosition );
```

3.7.2 Computing All Links' Elasticity associated with Dominant Eigenvalue Values and Related Checkpoints

To compute the all links' elasticity values associated with dominant eigenvalue; the following steps would be repeated for every link in the compact version of the model, i.e. for every non-zero value in the compact gain matrix. Also, the calculations of the related checkpoints files are done in the same process (checkpoints calculations code lines are omitted from this section, the interested reader should refer to the appendices).

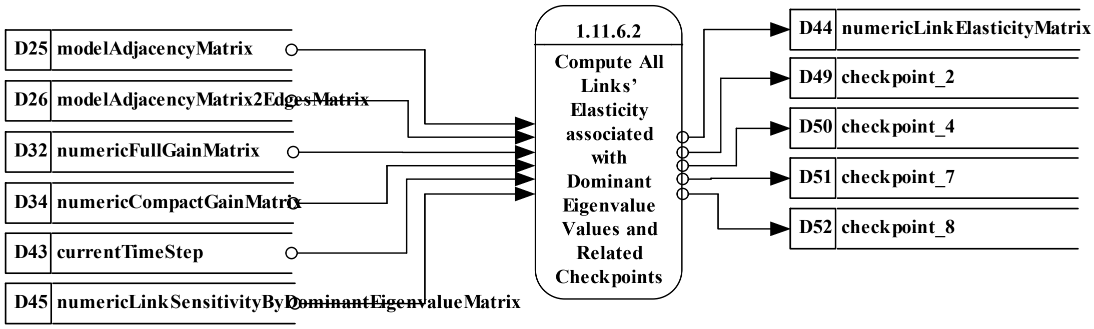


Figure 52: The DFD Level Four: Computing All Links' Elasticity associated with Dominant Eigenvalue Values and Related Checkpoints

```
% Computing All Links' Elasticity associated with Dominant Eigenvalue
Values
numericElasticityMatrix =
numericLinkSensitivityByDominantEigenvalueMatrix .*
numericCompactGainMatrix;
% The Full Elasticity Values Matrix
[ x , y ] = find( numericCompactGainMatrix ~= 0 );
for I = 1 : length( x ),
    ...
end
```

3.7.2.1Finding All Paths between Two Variables in the Compact Model

The *computeLinkElasticity* function uses *allpathn* function to find all paths between two variables in the compact model (a level and a net-flow), and then removes paths that pass through other levels to avoid redundancy computation error.

For more information the reader should refer to the "Digraph toolbox" (Bahar, M.; Jantzen, J., 1995).

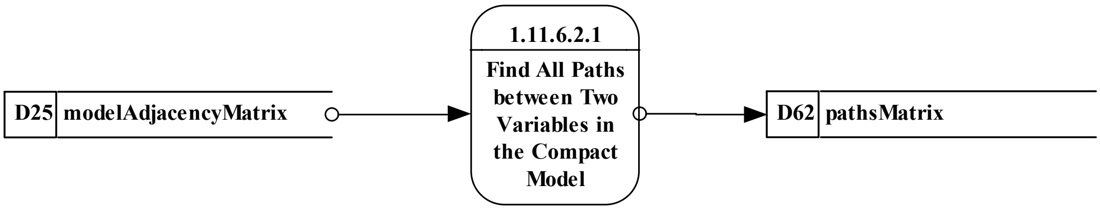


Figure 53: The DFD Level Five: Finding All Paths between Two Variables in the Compact Model

```
% Finding All Paths between Two Variables in the Compact Model
pathsMatrix = allpathn( y( I ) , x( I ) , modelAdjacencyMatrix );
% Deleting paths that pass through a level
w = [];
for K = 1 : size( pathsMatrix , 1 ) ,
    path = nonzeros( pathsMatrix( K , : ) );
    if any( path( 2 : end - 1 ) <= numLevels ),
        w = [ w K ];
    end
end
pathsMatrix( w , : ) = [];
```

3.7.2.2Computing Gain and Dominant Eigenvalue Elasticity Values of the k^{th} Path

After finding the paths, the *computeLinkElasticity* function uses *computePathsGain* function to compute the gain of these paths.

For more information the reader should refer to *computePathsGain* section.

The elasticity value of any path is computed by the following relation:

$$E_P = g_P \frac{s}{\lambda}$$

Where:

E_P : The dominant eigenvalue elasticity values of the path P.

g_P: The gain of the path P.

$\frac{s}{\lambda}$: The ratio of the path sensitivity to the dominant eigenvalue s to the dominant eigenvalue λ.

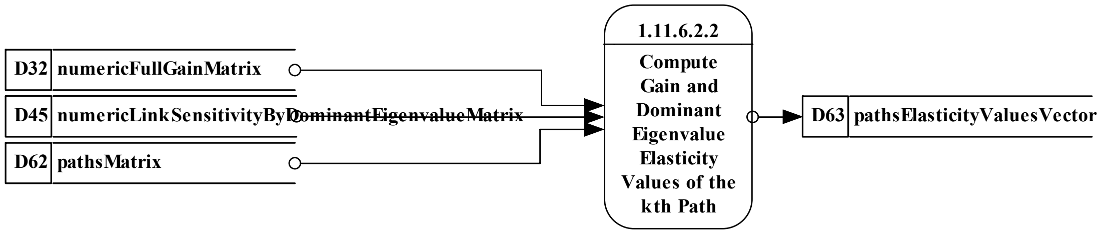


Figure 54: The DFD Level Five: Computing Gain and Dominant Eigenvalue Elasticity Values of the k^th Path

```
% Computing Gain and Dominant Eigenvalue Elasticity Values of the kth
Path
pathsGainsVector = computePathsGain( numericFullGainMatrix ,
pathsMatrix );
% the Elasticity value of the kth path
pathsElasticityValuesVector = pathsGainsVector *
numericLinkSensitivityByDominantEigenvalueMatrix( x( I ) , y( I ) );
```

3.7.2.3Computing Gain and Dominant Eigenvalue Elasticity Values of the Elements of the k^{th} Path

Every path in the compact model is corresponding to one or more paths in the full model, and in the last step the function has computed the gain and dominant eigenvalue elasticity of the path in the compact model. What is needed now is to divide this dominant eigenvalue over the corresponding paths, since their gains could be computed easily using *computePathsGain2* function, this function performs the following:

- Gets one of the paths out of the paths matrix *pathsMatrix* and removes the padding zeros

- Adds the dominant eigenvalue elasticity of that path to the dominant eigenvalue elasticity of every link in that path

$$E_\ell = \sum_{P \supset \{\ell\}} E_P$$

Where:

E_ℓ : Dominant eigenvalue elasticity of link ℓ which is contained in the path P.

- Computes the ratio $\frac{s_\ell}{\lambda}$ for every link in that path

$$\frac{s_\ell}{\lambda} = E_\ell \frac{1}{\sum_{P \supset \{\ell\}} g_P}$$

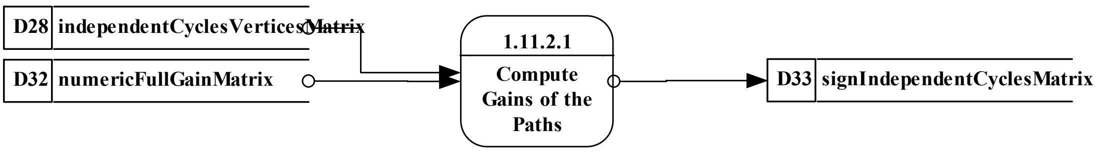


Figure 55: The DFD Level Four: Computing Gains the Paths

```
% Computing Gain and Dominant Eigenvalue Elasticity Values of the
Elements of the kth Path
for K = 1 : size( pathsMatrix , 1 ) ,
    % the kth path from y(I) to x(I)
    path = nonzeros( pathsMatrix( K , : ) );
    % for each element in the path
    for J = 1 : length( path ) - 1 ,
        numericLinkElasticityVector(
modelAdjacencyMatrix2EdgesMatrix( path( J+1 ) , path( J ) ) ) =
numericLinkElasticityVector( modelAdjacencyMatrix2EdgesMatrix( path(
J+1 ) , path( J ) ) ) + pathsElasticityValuesVector(K);
        numericFullElasticityMatrix( path( J+1 ) , path( J ) ) =
numericFullElasticityMatrix( path( J+1 ) , path( J ) ) +
pathsElasticityValuesVector( K );
        tempPathGain = computePathsGain2( numericFullGainMatrix ,
path , path( J+1 ) , path( J ) );
         tempPathSensitivityByDominantEigenvalueMatrix = tempPathGain
* numericLinkSensitivityByDominantEigenvalueMatrix( x( I ) , y( I )
```

```
);
        numericFullSensitivityByDominantEigenvalueMatrix( path( J+1 )
, path( J ) ) = numericFullSensitivityByDominantEigenvalueMatrix(
path( J+1 ) , path( J ) ) +
tempPathSensitivityByDominantEigenvalueMatrix;
         numericLinkSensitivityByDominantEigenvalueVector(
modelAdjacencyMatrix2EdgesMatrix( path( J+1 ) , path( J ) ) ) =
umericLinkSensitivityByDominantEigenvalueVector(
modelAdjacencyMatrix2EdgesMatrix( path( J+1 ) , path( J ) ) ) +
tempPathSensitivityByDominantEigenvalueMatrix;
    end
End
```

3.8 The computeInputElasticity Function

The function computes the dominant eigenvalue elasticity values of the inputs.

3.8.1 Computing Inputs' Elasticity Values Associated with the Dominant Eigenvalue

The function computes the dominant eigenvalue elasticity of the inputs one at a time; it goes into a for-loop with rounds number equals to the number of inputs, and every round it performs this calculation according to the following equation:

$$E_u = \frac{1}{\lambda}\left(\sum_{r=1}^{N_\ell} s_{\ell_r} \frac{\partial \ell_r}{\partial u}\right) u$$

Where:

E_u: The dominant eigenvalue elasticity value of the input u

$\frac{\partial \ell_r}{\partial u}$: Element of the *numericLinkGain2InputJacobianMatrix*

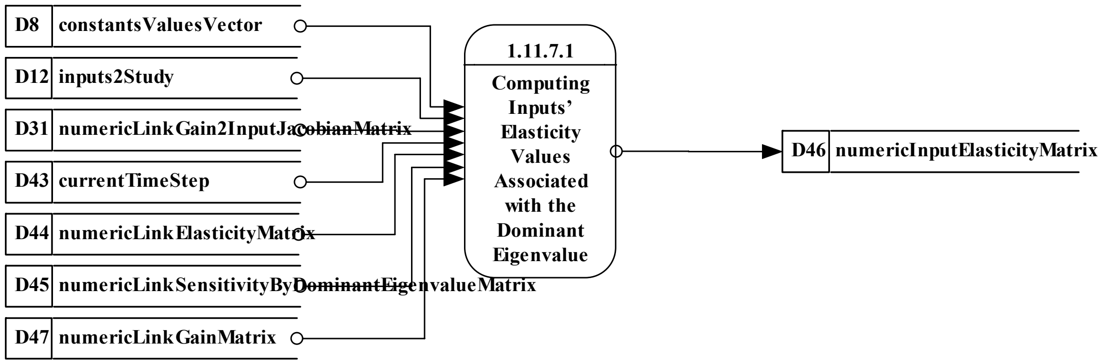


Figure 56: The DFD Level Four: Computing Inputs' Elasticity Values Associated with the Dominant Eigenvalue

```
% Computing Inputs' Elasticity Values Associated with the Dominant
Eigenvalue
for I = 1 : numInputs,
    numericInputElasticityVector( I ) = constantsValuesVector( I ) *
sum( numericLinkGain2InputJacobianMatrix( : , I ) .*
numericLinkSensitivityByDominantEigenvalueVector );
end
```

3.9 The computeIndependentCycleElasticity Function

The function computes the independent cycles (loops) elasticity values matrix associated with dominant eigenvalue.

3.9.1 Computing Linearly Independent Loops' Elasticity Values Associated with the Dominant Eigenvalue

The function computes the dominant eigenvalue elasticity of all linearly independent loops one time using the matrix form; it performs this calculation according to the following equation:

$$\mathbf{E}_{\ell} = \mathbf{C}_{r}\mathbf{E}_{\kappa}$$

Where:

$\mathbf{E}_{\ell}$: The vector of dominant eigenvalue elasticity values of the link ℓ

$\mathbf{E}_{\kappa}$: The vector of dominant eigenvalue elasticity values of the loop κ

The reduced cycles' matrix $\mathbf{C}_r$ is not a square matrix so that it can not be inversed and the last equation can not be solved using Cramer's, although it can be rewritten as:

$$\mathbf{E}_\kappa = \frac{\mathbf{C}_r}{\mathbf{E}_\ell}$$

And Matlab can find solution and compute $\mathbf{E}_\kappa$ using a least squares technique.

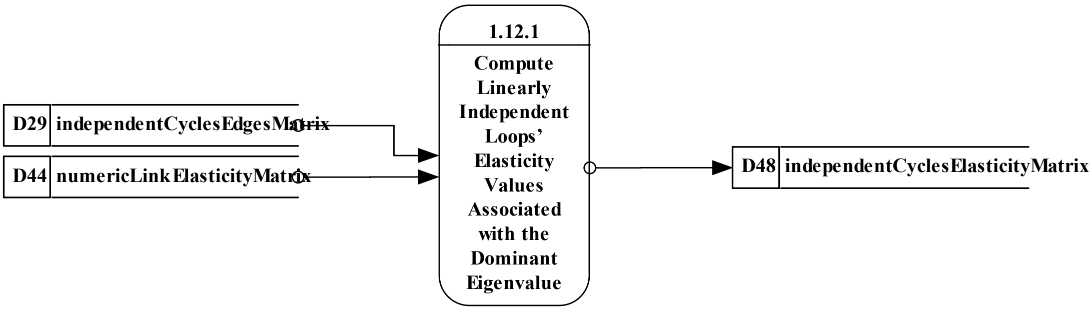


Figure 57: The DFD Level Three: Computing Linearly Independent Loops' Elasticity Values Associated with the Dominant Eigenvalue

```
% Computing Linearly Independent Loops' Elasticity Values Associated
with the Dominant Eigenvalue
% [ k1 ]         [ l1 ]
% [ k2 ]         [ l2 ]
% [ .  ] = Cr * [ .  ]
% [ .  ]         [ lm ]
% [ kn ]
%
% k: links   ,   l: loops
Cr = independentCyclesEdgesMatrix.';
independentCyclesElasticityMatrix = Cr \ numericLinkElasticityMatrix;
```

3.10 The printOutputs Function

The *printOutputs* function prints the outputs of the analysis function into a file specified by the user.

3.10.1 Printing All Eigenvalues and Their Dominance Percentage

The function performs the following steps:

- Prints a title for this section with the name of the level to study
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of eigenvalues in order to print the values of all eigenvalues as well as percentage of their contributions
- Prints a line in order to separate this section from the next section

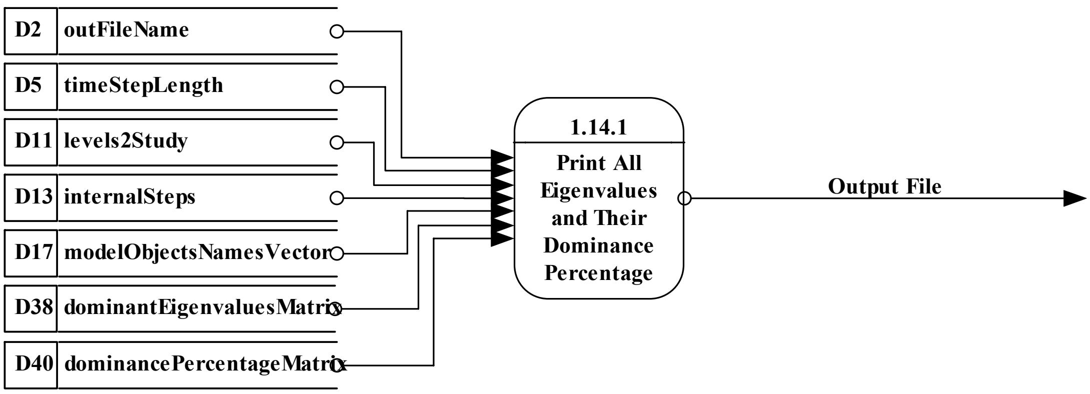


Figure 58: The DFD Level Three: Printing All Eigenvalues and Their Dominance Percentage

```
% Printing All Eigenvalues and Their Dominance Percentage
fwrite( fid , [ 'The eigenvalues and their dominance percentage
contribution to the level variable ''' char( modelObjectsNamesVector(
levels2Study ) ) ''':' ] );
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    for J = 1 : length( dominantEigenvaluesMatrix( I , : ) ),
        fwrite( fid , [ num2str(  dominantEigenvaluesMatrix( I , J )
) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I , J ) ) '%.' ] );
```

```
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.2 Identifying Links of the Model

The function identifies the links of the model by identifying the non-zero values in the full gain matrix, and then saves their indices into two separate vectors; one for the rows and one for the columns, to use them in the following sections.

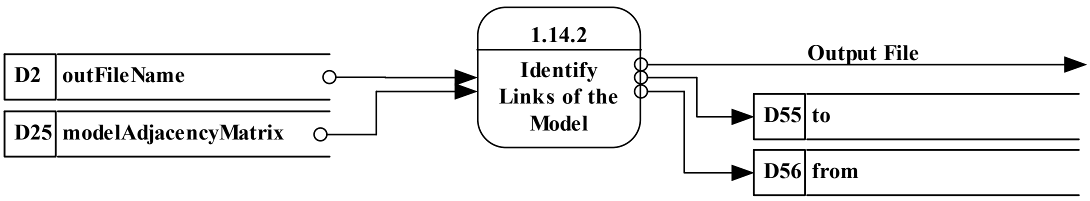


Figure 59: The DFD Level Three: Identifying Links of the Model

```
% Finding Links of the Model
[ to , from ] = find( modelAdjacencyMatrix );
```

3.10.3 Printing Links' Gains

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately

- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of eigenvalues in order to print the values of all eigenvalues as well as percentage of their contributions
- Prints a line in order to separate this section from the next section

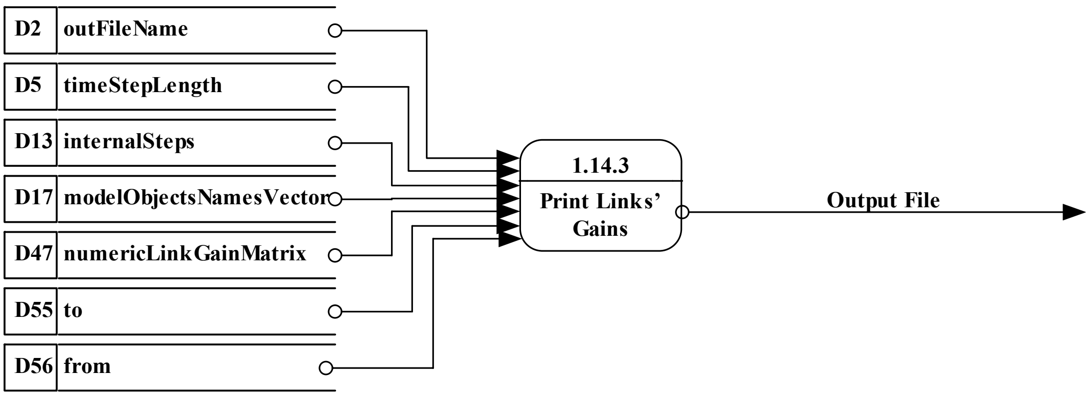


Figure 60: The DFD Level Three: Printing Links' Gains

```
% Printing Links' Gains
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All links and their gains:' ] );
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    for J = 1 : size( numericLinkGainMatrix , 1 ) ,
        fwrite( fid , [ char( modelObjectsNamesVector( from( J ) ) )
' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '  num2str( (
numericLinkGainMatrix( J , I ) ) ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'________________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.4 Printing Links' Dominant Eigenvalue Elasticity Values

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of links in order to print the names of all links as well as their corresponding dominant eigenvalue elasticity value
- Meanwhile, it saves the links' names and the values of links' gains effect on the real and imaginary parts of the dominant eigenvalue into two separate vector to be used in the next section
- Prints a line in order to separate this section from the next section

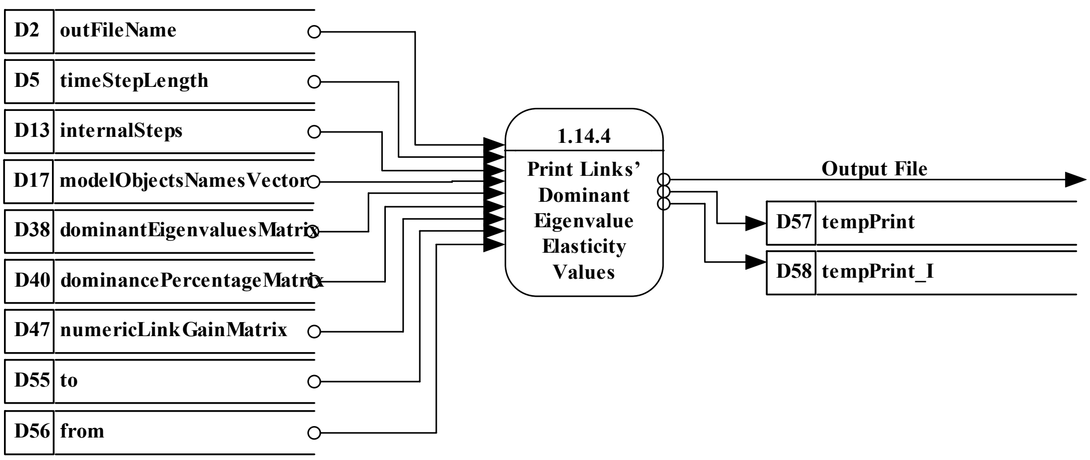


Figure 61: The DFD Level Three: Printing Links' Dominant Eigenvalue Elasticity Values

```
% Printing Links' Dominant Eigenvalue Elasticity Values
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All links and their elasticity values to the
dominant eigenvalue:' ] );
tempPrint = {};
```

```
tempPrint_I = {};
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '___________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if isreal( dominantEigenvaluesMatrix( I ) ),
        for J = 1 : size( numericLinkElasticityMatrix , 1 ) ,
            fwrite( fid , [ char( modelObjectsNamesVector( from( J )
) ) ' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '  num2str(
real( numericLinkElasticityMatrix( J , I ) ) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
            tempPrint{ J , I } = [ char( modelObjectsNamesVector(
from( J ) ) ) ' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '
num2str( real( numericLinkElasticityMatrix( J , I ) ) ) ];
            dummy_linkElasticity_Sorted( J , I ) = real(
numericLinkElasticityMatrix( J , I ) );
        end
    else
        for J = 1 : size( numericLinkElasticityMatrix , 1 ) ,
            fwrite( fid , [ char( modelObjectsNamesVector( from( J )
) ) ' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '  num2str(
( numericLinkElasticityMatrix( J , I ) ) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
            tempPrint{ J , I } = [ char( modelObjectsNamesVector(
from( J ) ) ) ' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '
num2str( real( numericLinkElasticityMatrix( J , I ) *
dominantEigenvaluesMatrix( I ) / abs( dominantEigenvaluesMatrix( I )
) ) ) ];
            tempPrint_I{ J , I } = [ char( modelObjectsNamesVector(
from( J ) ) ) ' --> ' char( modelObjectsNamesVector( to( J ) ) ) ': '
num2str( imag( numericLinkElasticityMatrix( J , I ) *
dominantEigenvaluesMatrix( I ) / abs( dominantEigenvaluesMatrix( I )
) ) ) ];
            dummy_linkElasticity_Sorted( J , I ) = real(
numericLinkElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I )
/ abs( dominantEigenvaluesMatrix( I ) ) );
            dummy_linkElasticity_Sorted_I( J , I ) = imag(
```

```
numericLinkElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I )
/ abs( dominantEigenvaluesMatrix( I ) ) );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.5 Printing Links' Dominant Eigenvalue Elasticity Values (Sorted)

The function performs the following steps:

- Sorts the elements of each of the two vectors that contains the values of links' gains effect on the real and imaginary parts of the dominant eigenvalue from the last section
- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of links in order to print the names of all links as well as their corresponding dominant eigenvalue elasticity value
- Prints a line in order to separate this section from the next section

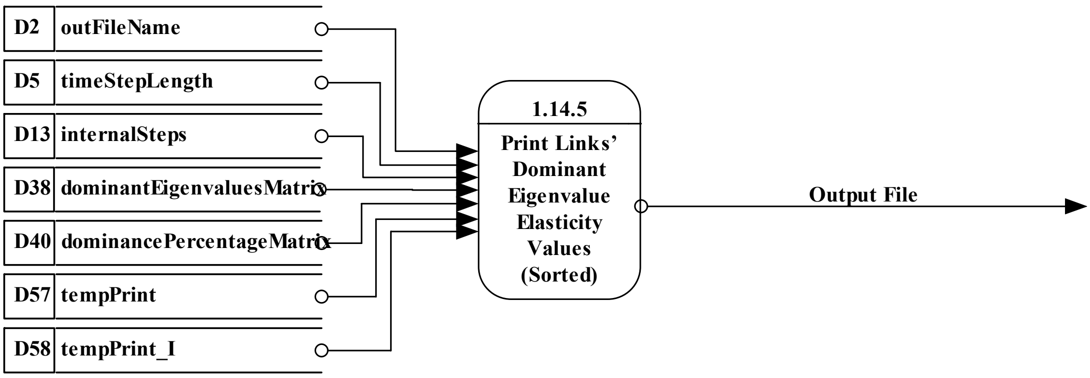


Figure 62: The DFD Level Three: Printing Links' Dominant Eigenvalue Elasticity Values (Sorted)

```
% Printing Links' Dominant Eigenvalue Elasticity Values (Sorted)
[ dummy_linkElasticity_Sorted , ...
  IX ] = sort( dummy_linkElasticity_Sorted , 1 );
[ dummy_linkElasticity_Sorted_I , ...
  IX_I ] = sort( dummy_linkElasticity_Sorted_I , 1 );
IX = flipud( IX );
IX_I = flipud( IX_I );
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All Links and their elasticity values to the
dominant eigenvalue (Sorted):' ] );
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ 'Effect on the Envelope:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
    for J = 1:size( numericLinkElasticityMatrix , 1 ) ,
        fwrite( fid , tempPrint{ IX( J , I ) , I } );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
```

```
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ 'Effect on the Frequency:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        for J = 1:size( numericLinkElasticityMatrix , 1 ) ,
            fwrite( fid , tempPrint_I{ IX_I( J , I ) , I } );
            fwrite( fid , [ sprintf( '\n' ) ] );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.6 Printing Inputs' Dominant Eigenvalue Elasticity Values

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of inputs in order to print the names of all inputs as well as their corresponding dominant eigenvalue elasticity value
- Meanwhile, it saves the inputs' names and the values of inputs' gains effect on the real and imaginary parts of the dominant eigenvalue into two separate vector to be used in the next section
- Prints a line in order to separate this section from the next section

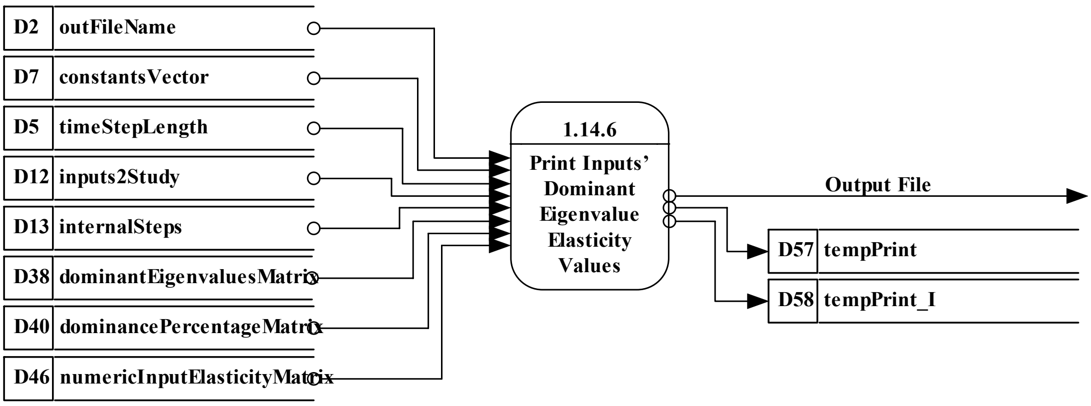


Figure 63: The DFD Level Three: Printing Inputs' Dominant Eigenvalue Elasticity Values

```
% Printing Inputs' Dominant Eigenvalue Elasticity Values
dummy_InputElasticity_Sorted = zeros( size(
numericInputElasticityMatrix ) );
dummy_InputElasticity_Sorted_I = zeros( size(
numericInputElasticityMatrix ) );
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All inputs and their elasticity values to the
dominant eigenvalue:' ] );
tempPrint = {};
tempPrint_I = {};
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if isreal( dominantEigenvaluesMatrix( I ) ),
        for J = 1 : length( inputs2Study  ),
            fwrite( fid , [ char( constantsVector( inputs2Study( J )
) ) ': ' num2str( real( numericInputElasticityMatrix( J , I ) ) ) ]
);
            fwrite( fid , [ sprintf( '\n' ) ] );
            tempPrint{ J , I } = [ char( constantsVector(
inputs2Study( J ) ) ) ': ' num2str( real(
numericInputElasticityMatrix( J , I ) ) ) ];
```

```
            dummy_InputElasticity_Sorted( J , I ) = real(
numericInputElasticityMatrix( J , I ) );
        end
    else
        for J = 1 : length( inputs2Study  ),
            fwrite( fid , [ char( constantsVector( inputs2Study( J )
) ) ': ' num2str( ( numericInputElasticityMatrix( J , I ) ) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
            tempPrint{ J , I } = [ char( constantsVector(
inputs2Study( J ) ) ) ': ' num2str( real(
numericInputElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I
) / abs( dominantEigenvaluesMatrix( I ) ) ) ) ];
            tempPrint_I{ J , I } = [ char( constantsVector(
inputs2Study( J ) ) ) ': ' num2str( imag(
numericInputElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I
) / abs( dominantEigenvaluesMatrix( I ) ) ) ) ];
            dummy_InputElasticity_Sorted( J , I ) = real(
numericInputElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I
) / abs( dominantEigenvaluesMatrix( I ) ) );
            dummy_InputElasticity_Sorted_I( J , I ) = imag(
numericInputElasticityMatrix( J , I ) * dominantEigenvaluesMatrix( I
) / abs( dominantEigenvaluesMatrix( I ) ) );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.7 Printing Inputs' Dominant Eigenvalue Elasticity Values (Sorted)

The function performs the following steps:

- Sorts the elements of each of the two vectors that contains the values of inputs' gains effect on the real and imaginary parts of the dominant eigenvalue from the last section
- Prints a title for this section

- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of inputs in order to print the names of all inputs as well as their corresponding dominant eigenvalue elasticity value
- Prints a line in order to separate this section from the next section

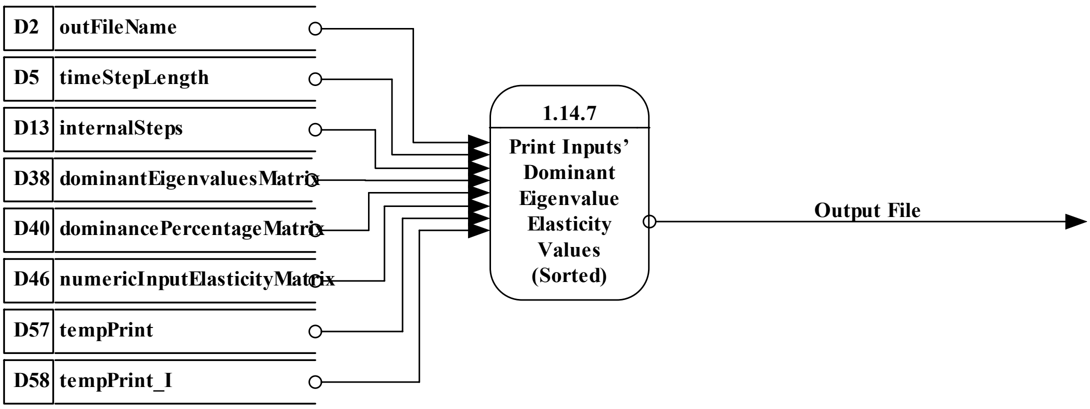


Figure 64: The DFD Level Three: Printing Inputs' Dominant Eigenvalue Elasticity Values (Sorted)

```
% Printing Inputs' Dominant Eigenvalue Elasticity Values (Sorted)
[ dummy_InputElasticity_Sorted , ...
  IX ] = sort( dummy_InputElasticity_Sorted , 1 );
[ dummy_InputElasticity_Sorted_I , ...
  IX_I ] = sort( dummy_InputElasticity_Sorted_I , 1 );
IX = flipud( IX );
IX_I = flipud( IX_I );
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All inputs and their elasticity values to the
dominant eigenvalue (Sorted):' ] );
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
```

```
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ 'Effect on the Envelope:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
    for J = 1:size( numericInputElasticityMatrix , 1 ) ,
        fwrite( fid , tempPrint{ IX( J , I ) , I } );  %IX(:,1)
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ 'Effect on the Frequency:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        for J = 1:size( numericInputElasticityMatrix , 1 ) ,
            fwrite( fid , tempPrint_I{ IX_I( J , I ) , I } );
            fwrite( fid , [ sprintf( '\n' ) ] );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.8 Printing All Loops

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the number of loops in order to print a tilted information section about each of these loops separately

- Inside the previous for-loop; it goes into another for-loop that has rounds equal to the number of links in the current loop in order to save these links inside temporary characters vector separated by ' --> ', and ends the inner for-loop.
- Prints temporary characters vector.
- Ends the first for-loop and prints a line in order to separate this section from the next section

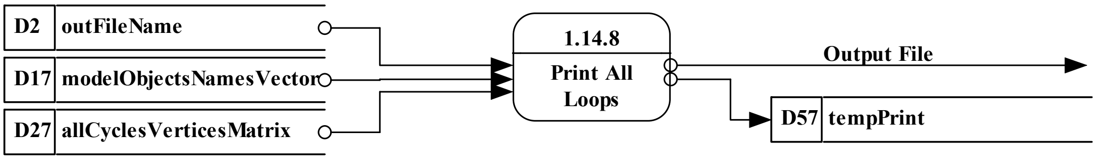


Figure 65: The DFD Level Three: Printing All Loops

```
% Printing All Loops
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'All loops:' ] );
for I = 1:size( allCyclesVerticesMatrix , 1 ) ,
    tempPrint = [];
    oneCycle = nonzeros( allCyclesVerticesMatrix( I , : ) ).';
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Loop ' num2str( I ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    for J = 1:size( oneCycle , 2 )-1 ,
        tempPrint = [ tempPrint , char( modelObjectsNamesVector(
oneCycle( J ) ) ) ];
        if J ~= size( oneCycle , 2 )-1 ,
            tempPrint = [ tempPrint , ' --> ' ];
        end
    end
    fwrite( fid , tempPrint );
    fwrite( fid , [ sprintf( '\n' ) ] );
end
fwrite( fid , [ sprintf( '\n' ) ] );
```

```
fwrite( fid , [
'___________________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.9 Printing User-selected Linearly Independent Loops

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the number of linearly independent loops in order to print a tilted information section about each of these loops separately
- Inside the previous for-loop; it goes into another for-loop that has rounds equal to the number of links in the current loop in order to print the links of that loop separated by ' --> '.
- Prints a line in order to separate this section from the next section

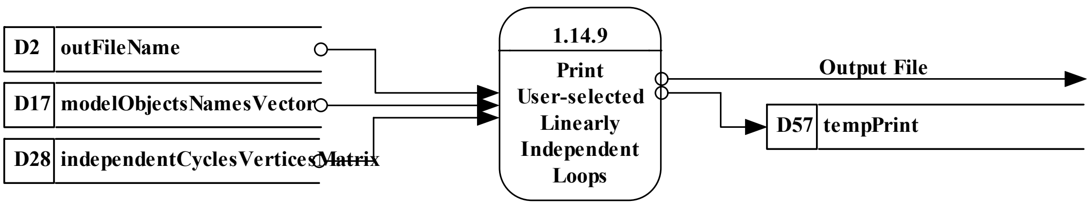


Figure 66: The DFD Level Three: Printing User-selected Linearly Independent Loops

```
% Printing User-selected Linearly Independent Loops
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'Linearly independent loops:' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
for I = 1:size( independentCyclesVerticesMatrix , 1 ) ,
    tempPrint = [];
    oneCycle = nonzeros( independentCyclesVerticesMatrix( I , : )
).';
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Loop ' num2str( I ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '____________________' ] );
```

```
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    for J = 1:size( oneCycle , 2 )-1 ,
        tempPrint = [ tempPrint , char( modelObjectsNamesVector(
oneCycle( J ) ) ) ];
        if J ~= size( oneCycle , 2 )-1 ,
            tempPrint = [ tempPrint , ' --> ' ];
        end
    end
    fwrite( fid , tempPrint );
    fwrite( fid , [ sprintf( '\n' ) ] );
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.10 Printing User-Selected Linearly Independent Loops' Dominant Eigenvalue Elasticity Values

The function performs the following steps:

- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of the user-selected linearly independent loops in order to print the names of all user-selected linearly independent loops as well as their corresponding dominant eigenvalue elasticity value
- Meanwhile, it saves the user-selected linearly independent loops' names and the values of links' gains effect on the real and imaginary parts of the dominant eigenvalue into two separate vector to be used in the next section

- Prints a line in order to separate this section from the next section

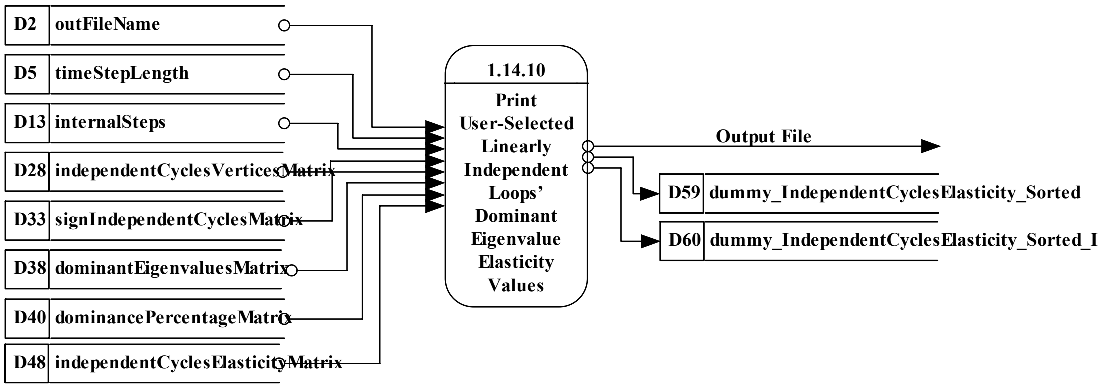


Figure 67: The DFD Level Three: Printing User-Selected Linearly Independent Loops' Dominant Eigenvalue Elasticity Values

```
% Printing User-Selected Linearly Independent Loops' Dominant
Eigenvalue Elasticity Values
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'Independent loops'' elasticity values:' ] );
independentCyclesElasticityMatrix =
independentCyclesElasticityMatrix.';
dummy_IndependentCyclesElasticity_Sorted = zeros( size(
independentCyclesElasticityMatrix ) );
dummy_IndependentCyclesElasticity_Sorted_I = zeros( size(
independentCyclesElasticityMatrix ) );
for I = internalSteps ,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if isreal( dominantEigenvaluesMatrix( I ) ),
        for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
            fwrite( fid , [ 'Loop ' num2str( K ) ' (Polarity: '
num2str( signIndependentCyclesMatrix( I , K ) ) '): ' num2str( real(
independentCyclesElasticityMatrix( I , K ) ) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
            dummy_IndependentCyclesElasticity_Sorted( I , K ) = real(
```

```
independentCyclesElasticityMatrix( I , K ) );
        end
    else
        for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
            fwrite( fid , [ 'Loop ' num2str( K ) ' (Polarity: '
num2str( signIndependentCyclesMatrix( I , K ) ) '): ' num2str(
independentCyclesElasticityMatrix( I , K ) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
            dummy_IndependentCyclesElasticity_Sorted( I , K ) = real(
independentCyclesElasticityMatrix( I , K ) *
dominantEigenvaluesMatrix( I ) / abs( dominantEigenvaluesMatrix( I )
) );
            dummy_IndependentCyclesElasticity_Sorted_I( I , K ) =
imag( independentCyclesElasticityMatrix( I , K ) *
dominantEigenvaluesMatrix( I ) / abs( dominantEigenvaluesMatrix( I )
) );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.10.11 Printing User-Selected Linearly Independent Loops' Dominant Eigenvalue Elasticity Values (Sorted)

The function performs the following steps:

- Sorts the elements of each of the two vectors that contains the values of user-selected linearly independent loops' gains effect on the real and imaginary parts of the dominant eigenvalue from the last section
- Prints a title for this section
- Goes into a for-loop that has rounds equal to the user-selected time steps for analysis in order to print a tilted information section about each of these steps separately

- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of the user-selected linearly independent loops in order to print the names of all user-selected linearly independent loops as well as their corresponding dominant eigenvalue elasticity value
- Prints a line in order to separate this section from the next section

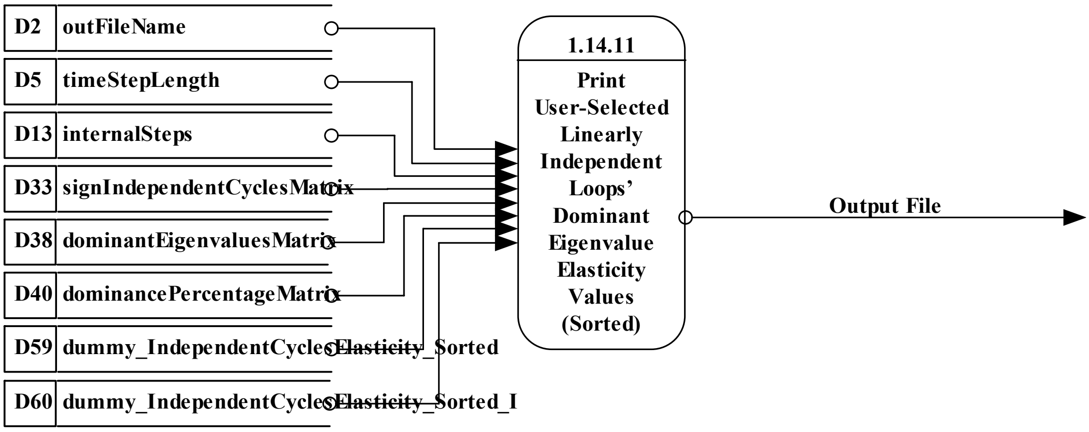


Figure 68: The DFD Level Three: Printing User-Selected Linearly Independent Loops' Dominant Eigenvalue Elasticity Values (Sorted)

```
% Printing User-Selected Linearly Independent Loops' Dominant
Eigenvalue Elasticity Values (Sorted)
[ dummy_IndependentCyclesElasticity_Sorted , ...
  IX ] = sort( dummy_IndependentCyclesElasticity_Sorted , 2 );
[ dummy_IndependentCyclesElasticity_Sorted_I , ...
  IX_I ] = sort( dummy_IndependentCyclesElasticity_Sorted_I , 2 );
dummy_IndependentCyclesElasticity_Sorted = fliplr(
dummy_IndependentCyclesElasticity_Sorted );
dummy_IndependentCyclesElasticity_Sorted_I = fliplr(
dummy_IndependentCyclesElasticity_Sorted_I );
IX = fliplr( IX );
IX_I = fliplr( IX_I );
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [ 'Independent loops'' elasticity values (Sorted):' ]
);
for I = internalSteps,
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) *
timeStepLength ) ':' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
```

```
    fwrite( fid , [ '__________________' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix( I ) ) ', with percentage contribution: '
int2str( dominancePercentageMatrix( I ) ) '%.' ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    fwrite( fid , [ sprintf( '\n' ) ] );
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ 'Effect on the Envelope:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
    for K = 1:size( independentCyclesVerticesMatrix , 1 ),
        fwrite( fid , [ 'Loop ' num2str( IX( I , K ) ) ' (Polarity: '
num2str( signIndependentCyclesMatrix( I , IX( I , K ) ) ) '): '
num2str( dummy_IndependentCyclesElasticity_Sorted( I , K ) ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
    end
    if ~isreal( dominantEigenvaluesMatrix( I ) ),
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ 'Effect on the Frequency:' ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        fwrite( fid , [ sprintf( '\n' ) ] );
        for K = 1:size( independentCyclesVerticesMatrix , 1 ),
            fwrite( fid , [ 'Loop ' num2str( IX_I( I , K ) ) '
(Polarity: ' num2str( signIndependentCyclesMatrix( I , IX_I( I , K )
) ) '): ' num2str( dummy_IndependentCyclesElasticity_Sorted_I( I , K
) ) ] );
            fwrite( fid , [ sprintf( '\n' ) ] );
        end
    end
end
fwrite( fid , [ sprintf( '\n' ) ] );
fwrite( fid , [
'____________________________________________________' ] );
fwrite( fid , [ sprintf( '\n' ) ] );
```

3.11 The jac Function

The *jac* function computes Jacobian, according to the following equation:

$$jac(\mathbf{x},\mathbf{y}) = \frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \cdots & \frac{\partial x_1}{\partial y_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial y_1} & \cdots & \frac{\partial x_m}{\partial y_n} \end{bmatrix}$$

And to implement this; the function performs the following steps:

- Goes into a for-loop that has rounds equal to the number of elements in the first vector $\mathbf{x}$
- Goes into a for-loop that has rounds equal to the number of elements in the second vector $\mathbf{y}$
- Uses the function *differentiate* to differentiate all elements of $\mathbf{x}$ to $\mathbf{y}$

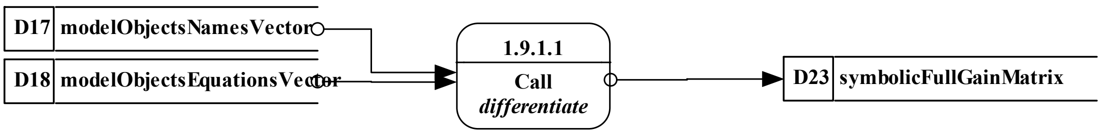


Figure 69: The DFD Level Four: Computing Jacobian by Calling Differentiate

```
% jac( x , y )
for I = 1 : length( x ),
    for J = 1 : length( y ),
        out( I , J ) = differentiate( x( I ) , y( J ) );
    end
end
```

3.12 The differentiate Function

The *differentiate* function computes partial differentiation, according to the following equation:

$$differentiate(x,y) = \frac{\partial x}{\partial y}$$

And to implement this; the function performs the following steps:

- Tests if the input x is *ifThenElse* function, and if so it differentiates both values of the *ifThenElse* to the input y by sending them to the symbolic toolbox, i.e. the if-true and if-false values, and returns another *ifThenElse*, with the same condition, but with the new differentiated if-true and if-false values
- Tests if the input x is *graph* function, and if so it replaces it with the *differentiateGraph* function with the same arguments of the *graph* function
- Sends all other x and y values to the symbolic toolbox to do the differentiation

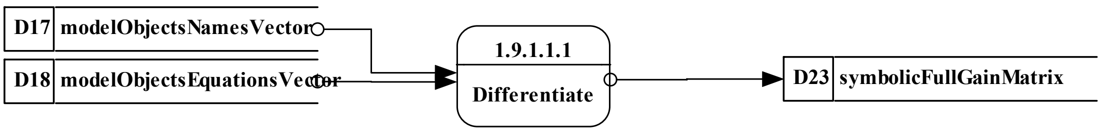


Figure 70: The DFD Level Five: Performing Differentiation

```
% differentiate( S , a )
str = char( S );
if strncmp( str , 'ifthenelse' , 10 ),
    ix1 = strfind( str , '(' );
    ix2 = strfind( str , ',' );
    ix3 = strfind( str , ')' );
    var0 = str( 1 : ix1( 1 ) - 1 );
    var1 = str( ix1( 1 ) + 1 : ix2( 1 ) - 1 );
    var2 = str( ix2( 1 ) + 1 : ix2( 2 ) - 1 );
    var3 = sym( str( ix2( 2 ) + 1 : ix2( 3 ) - 1 ) );
    var4 = sym( str( ix2( 3 ) + 1 : ix3( 1 ) - 1 ) );
    var3 = maple( 'map' , 'diff' , var3 , a );
    var4 = maple( 'map' , 'diff' , var4 , a );
    if var3 == sym( 0 ) & var4 == sym( 0 ),
        R = sym( 0 );
    else
        R = sym( [ var0 '(' var1 ',' var2 ',' char( var3 ) ',' char(
var4 ) ')' ] );
    end
```

```
elseif strncmp( str , 'graph' , 5 ),
    ix1 = strfind( str , '(' );
    ix2 = strfind( str , ',' );
    var1 = sym( str( ix1( 1 ) + 1 : ix2( 1 ) - 1 ) );
    var2 = maple( 'map' , 'diff' , var1 , a );
    R = sym( strrep( str , 'graph' , 'differentiateGraph' ) ) * var2;
else
    R = maple( 'map' , 'diff' , S , a );
end
```

3.13 The differentiateGraph Function

The *differentiateGraph* function computes differentiation of *graph* function, according to the following equation:

$$differentiateGraph(inp,x,y) = \frac{\partial\left(graph(inp,x,y)\right)}{\partial(inp)}$$

And to implement this; the function performs the following steps:

- Tests if the inputs of the original *graph* function inp, x and y are of a numeric values, if not it returns a symbolic expression $differentiateGraph(inp,x,y)$, else the following steps apply
- If the input x is not inside the range of the vector inp and on one of its sides, it returns slope of the line connecting the last two points in the inp vector from the related side
- If the input x is not inside the range of the vector inp and is not on one of its sides, it returns zero because the line after the range of the vector inp is parallel to the x-axis meaning zero slope
- If the input x is inside the range of the vector inp, and is between two points in the inp vector, it returns the of the slopes of the line connecting the two points enclosing the input x

- If the input x is inside the range of the vector inp, and is on one of the points in the inp vector, it returns the mean value of the slopes of the line connecting this point and the two points at each side in the inp vector

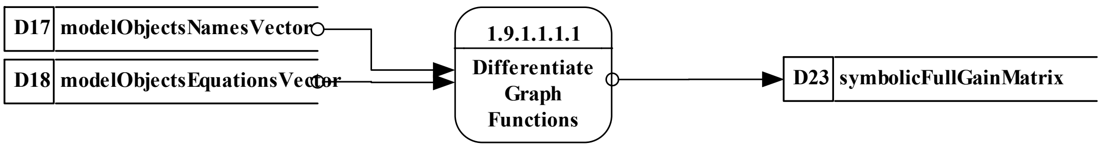


Figure 71: The DFD Level Six: Performing Differentiation to Graph Function

```
% differentiateGraph( inp , x , y )
n = (nargin - 1) / 2;
x = varargin( 1 : n );
y = varargin( n + 1 : end );
x = cell2num( x );
y = cell2num( y );
inp = subs( inp );
if isnumeric( inp ),
    [ x , IX ] = sort( x );
    y = y( IX );
    % Find indices of subintervals, x( k ) <= inp < x( k + 1 ),
    % or inp < x( 1 ) or inp >= x( end )
    k = sum( x < inp ); % 0 ---> n
    if k == 0,
        % Extrapolate
        if inp == x( 1 ),
            out = ( ( y( 2 ) - y( 1 ) ) / ( x( 2 ) - x( 1 ) ) ) / 2;
        else
            out = 0;
        end
    elseif k == n,
        % Extrapolate
        if inp == x( end ),
            out = ( ( y( end ) - y( end -1 ) ) / ( x( end ) - x( end
- 1 ) ) ) / 2;
        else
            out = 0;
        end
    else
        % Interpolate
```

```
        if inp == x( k ),
            out = mean( [ ( y( k + 1 ) - y( k ) ) / ( x( k + 1 ) - x( k ) ) , ( y( k ) - y( k - 1 ) ) / ( x( k ) - x( k - 1 ) ) ] );
        else
            out = ( y( k + 1 ) - y( k ) ) / ( x( k + 1 ) - x( k ) ) ;
        end
    end
else
    out = sym( [ 'differentiateGraph(' char( inp ) ',' rowv( x ) ',' rowv( y ) ')' ] );
end
```

3.14 The computePathsGain Function

The *computePathsGain* function computes the gains of set of paths, according to the following equation:

$$g_P = \prod_{\ell \in P} g_\ell$$

And to implement this; the function performs the following steps:

- Goes into a for-loop that has rounds equal to the number of paths
- Inside the previous for-loop; it goes into another for-loop that has steps equal to the number of links of the path that is in-turn
- Multiply the gains of the links utilizing the previous for-loop

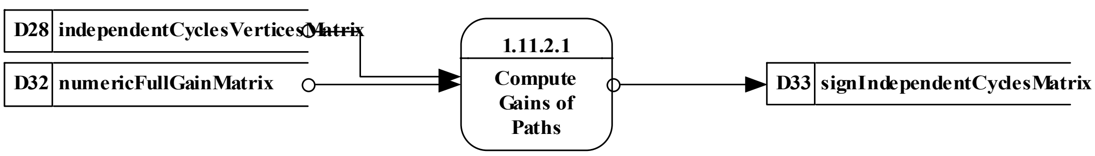


Figure 72: The DFD Level Four: Computing Gains of Paths (Related to Computing Polarity of the Linearly Independent Loops)

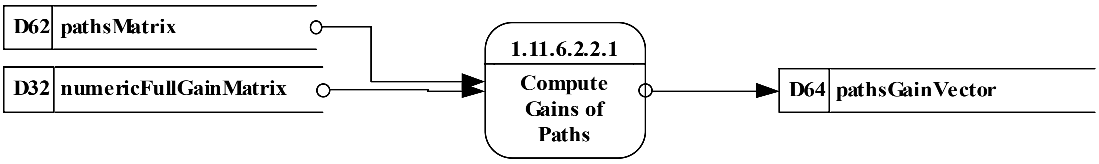


Figure 73: The DFD Level Six: Computing Gains of Paths (Related to Computing Gain and Dominant Eigenvalue Elasticity Values of the kth Path)

```
GV = ones( 1 , size( paths , 1 ) );
for i = 1 : size( paths , 1 ),
    cn = paths( i , 1 : max( find( paths( i , : ) ) ) );
    for j = 1 : length( cn ) - 1,
        GV( i ) = GV( i ) * G( cn( j + 1 ) , cn( j ) );
    end
end
```

3.15 The computePathsGain2 Function

The *computePathsGain2* function computes the gain of a path excluding the gain of a specific link that starts and ends at specific nodes, it is the same as the *computePathsGain* function, but it works only for only one path using the same equation of the *computePathsGain* function taking into consideration that it leaves the specific link and don't multiply it to others.

And to implement this; the function performs the following steps:

- Goes into a for-loop that has rounds equal to the number of links of the path
- Multiply the gains of the links utilizing the previous for-loop, excluding the link specified by the user

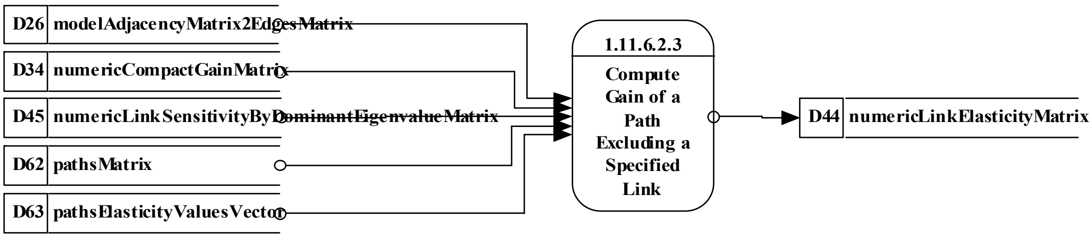


Figure 74: The DFD Level Five: Computing Gain of a Path Excluding a Specififed Link

```
GV = 1;
cn = path( 1 : max( find( path ) ) );
for j = 1 : length( cn ) - 1,
    if ~( ( cn( j + 1 ) == eNode ) & ( cn( j ) == sNode ) )
        GV = GV * G( cn( j+1 ),cn( j ) );
    end
end
```

Chapter 4

The Analysis Package: Application on the MGM

4.1 Introduction

This chapter applies the mathematical foundation of the eigenvalue analysis using the implemented Matlab code, to one of the system dynamics classical models. This chapter contains four sections other than this overview.

Section 4.2 and 4.3 give an overview about the market growth model taking into consideration its different sectors and important loops.

Section 4.4 declares the steps of the eigenvalue analysis of the behavior of the *Backlog* level variable of the market growth model as well as the resuts of the Analysis package.

Section 4.5 gives brief comments on the results of the Analysis package.

4.2 The Market Growth Model Overview

Basically Jay W. Forrester made his market growth model to show the feedback relationships frequently governing the growth of a new product in an open market. Despite the simplicity of the model, it exhibits a various interesting modes of behavior. The model consists of four main sectors, the Operations sector (relating orders to the resulting deliveries) the Salesmen sector (relating budget to the resulting salesmen), the Market sector (relating the delivery delay to the resulting sales effectiveness) and the Capacity sector (relating the ordering of production to the resulting production capacity). These sectors were originally coupled with switches to activate or deactivate one of them at a time. The full set of equations written for Powersim Studio, are listed in the appendix.

In this chapter, we will focus on the seventh run documented in Forrester's original paper – he called it: "Delivery Delay Goal Based on Past Performance". The run demonstrates the effects of operating with an eroding goal structure and is characterized by setting all of the four switches to 1:

The model, portrayed in figure 75, is characterized by three major feedback loops, one reinforcing loop and two balancing loops. The reinforcing loop causes salesmen to be hired (or fired) as a result of an increasing (or decreasing) budget, resulting from the deliveries caused by the orders generated by the salesmen. The first, upper, balancing loop causes a reduction (or increase) in orders generated as a result of a decrease (or increase) in the sales effectiveness in response to an increased (or decreased) in the observed delivery delay resulting from an increasing (pr decreasing) backlog caused by an order left unmatched (or matched) by the delivery capacity. The second, lower, balancing loop causes an increase (or decrease) in capacity in response to (no) capacity ordering resulting from the observation of an increase (or decrease) in the delivery delay, caused by insufficient (or sufficient) capacity.

Switch 1 = 1, *Switch 2* = 1, *Switch 3* = 1 and *Delivery Delay Weighting* = 1

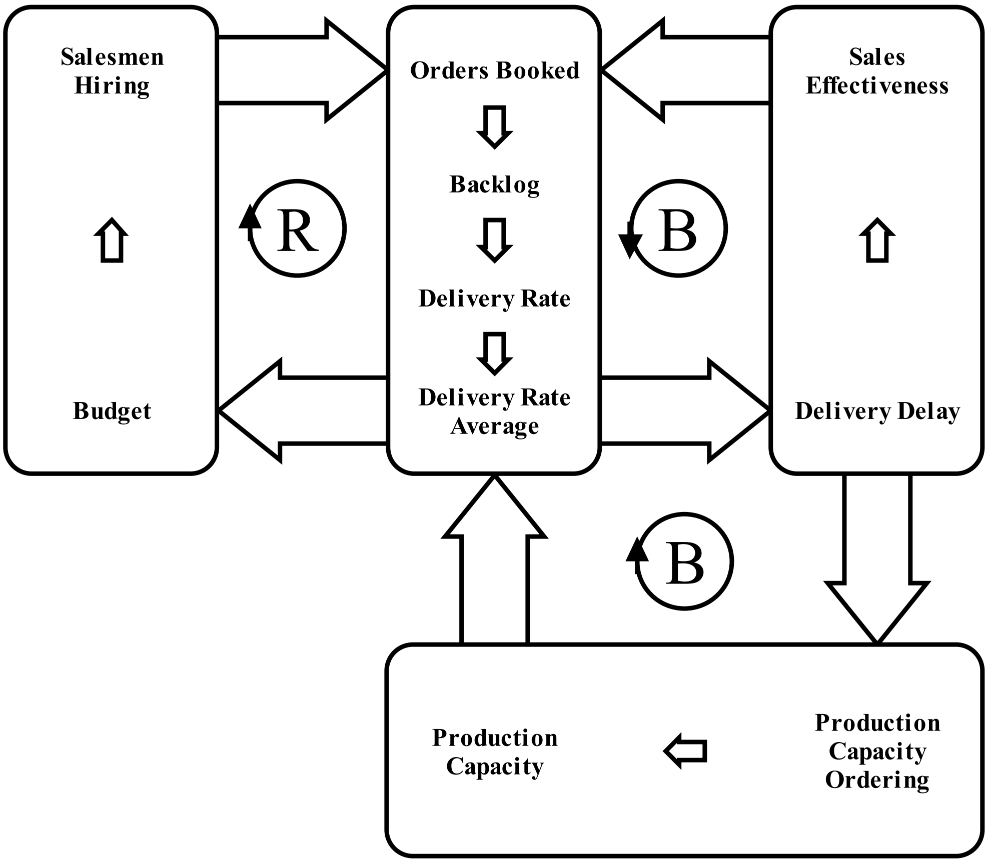


Figure 75: The Model structure including the Salesmen, Operations, Market and capacity sectors

4.3 The Market Growth Model Sectors

4.3.1 The Operations and Salesmen Sectors

Figure 76 shows the structure of the Operations and Salesmen sectors. It portrays the salesmen hiring process as well as the relationship between the orders booked generated by the salesmen and sales budget that allows for expansion, generated by the deliveries that results from the orders booked. These sectors incorporate two main loops[‡‡‡‡], a balancing and a reinforcing loop:

- *Salesmen* → *Salesmen Hired* → *Salesmen* **(B1)**
- *Backlog* → *Delivery Delay Minimum* → *Production Capacity Fraction* → *Delivery Rate* → *Delivery Rate Average Adjustment* → *Delivery Rate Average* → *Budget* → *Indicated Salesmen* → *Salesmen Hired* → *Salesmen* → *Salesmen Switch* → *Orders Booked* → *Backlog* **(R2)**

[‡‡‡‡] As identified by Jay W. Forrester in his original paper (Forrester, 1968).

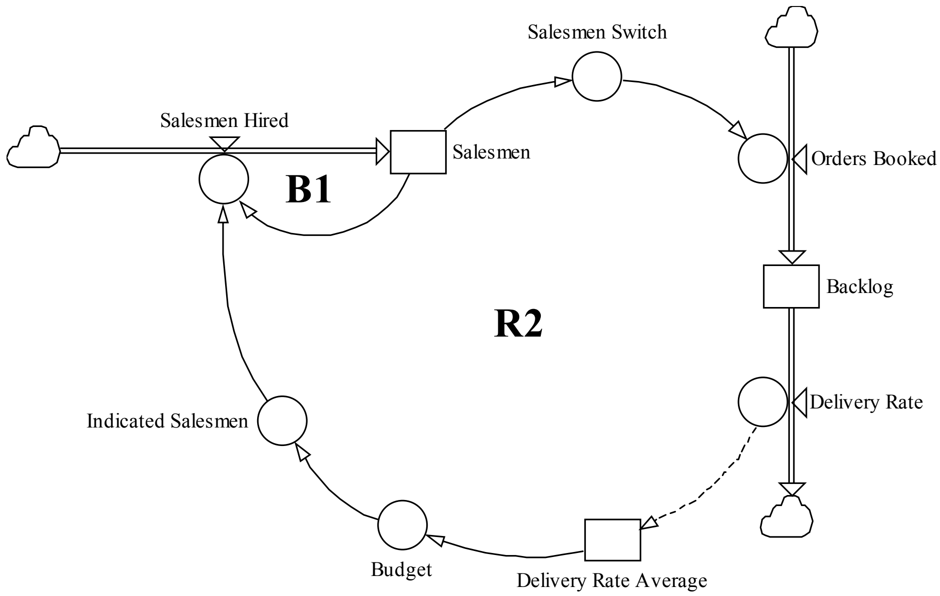


Figure 76: Salesmen-Hiring Sector

4.3.2 The Operations and Market Sectors

Figure 77 shows the structure of the Operations and Market sectors. These sectors determine both the product attractiveness and the consequent sales effectiveness resulting from delivery delay observed by the market, this delivery delay determined by the relationship between the rate of orders generated and the delivery rate governed by the capacity.

Also these sectors incorporates two main balancing loops[§§§§]:

- *Backlog* → *Delivery Delay Minimum* → *Production Capacity Fraction* → *Delivery Rate* → *Backlog* **(B3)**

- *Backlog* → *Delivery Delay Minimum* → *Production Capacity Fraction* → *Delivery Rate* → *Delivery Rate Average Adjustment* → *Delivery Rate Average* → *Delivery Delay Indicated* → *Delivery Delay Recognized By Company Adjustment* → *Delivery Delay Recognized By Company* → *Delivery Delay Recognized By Market Adjustment* → *Delivery Delay Recognized By Market* → *Sales Effectiveness From Delay Multiplier* → *Sales Effectiveness From Delay Switch* → *Sales Effectiveness* → *Orders Booked* → *Backlog* **(R4)**

[§§§§] As identified by Jay W. Forrester in his original paper (Forrester, 1968).

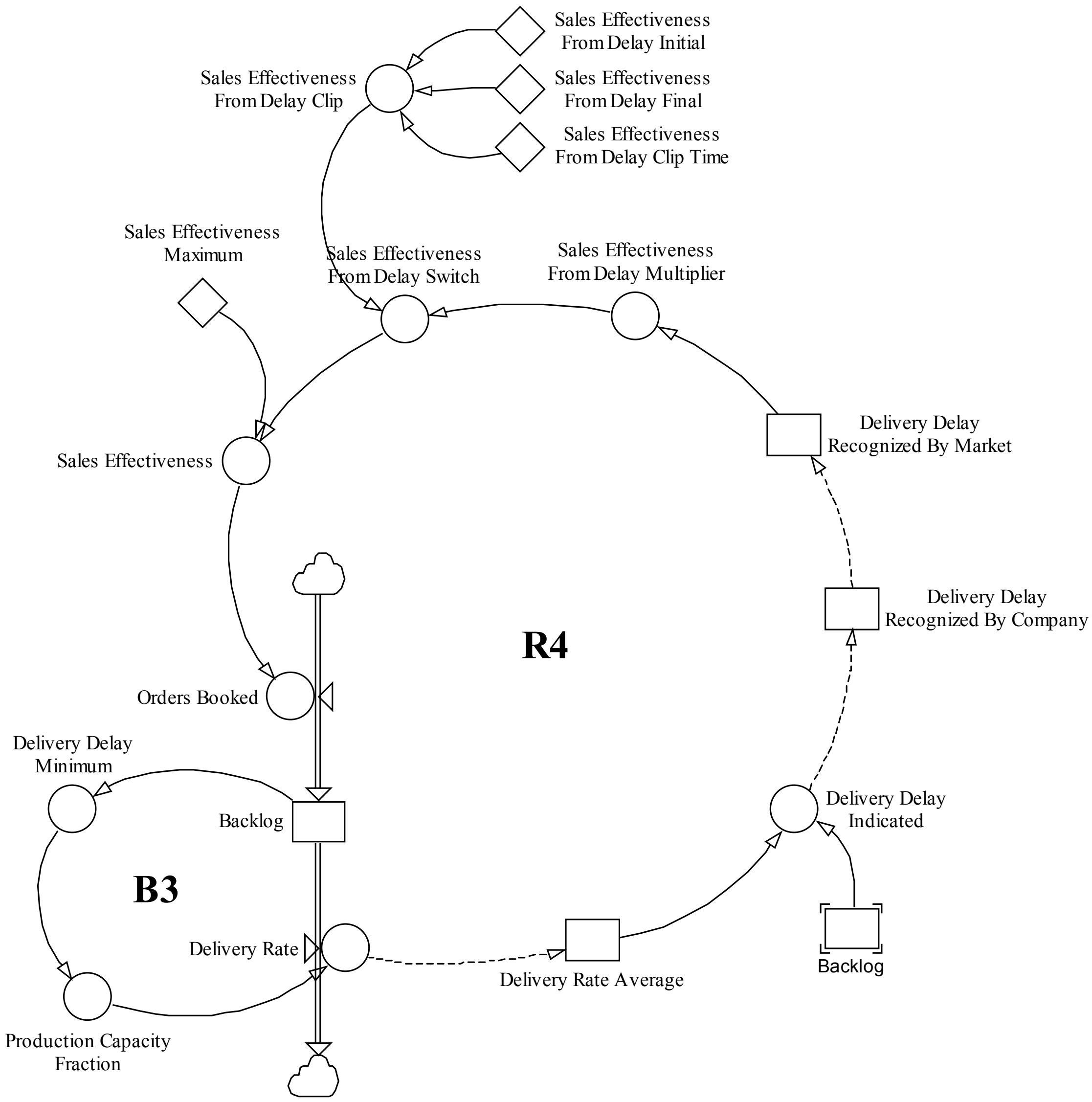


Figure 77: Market Sector

4.3.3 The Capacity Sector

Figure 78 shows the structure of the Capacity sector that responds to the delivery delay by providing additional production capacity in case the delivery delay increases and vice versa.

This sector incorporates only a single main reinforcing loop[*****]:

- *Production Capacity → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving → Production Capacity* **(R5)**

[*****] As identified by Jay W. Forrester in his original paper (Forrester, 1968).

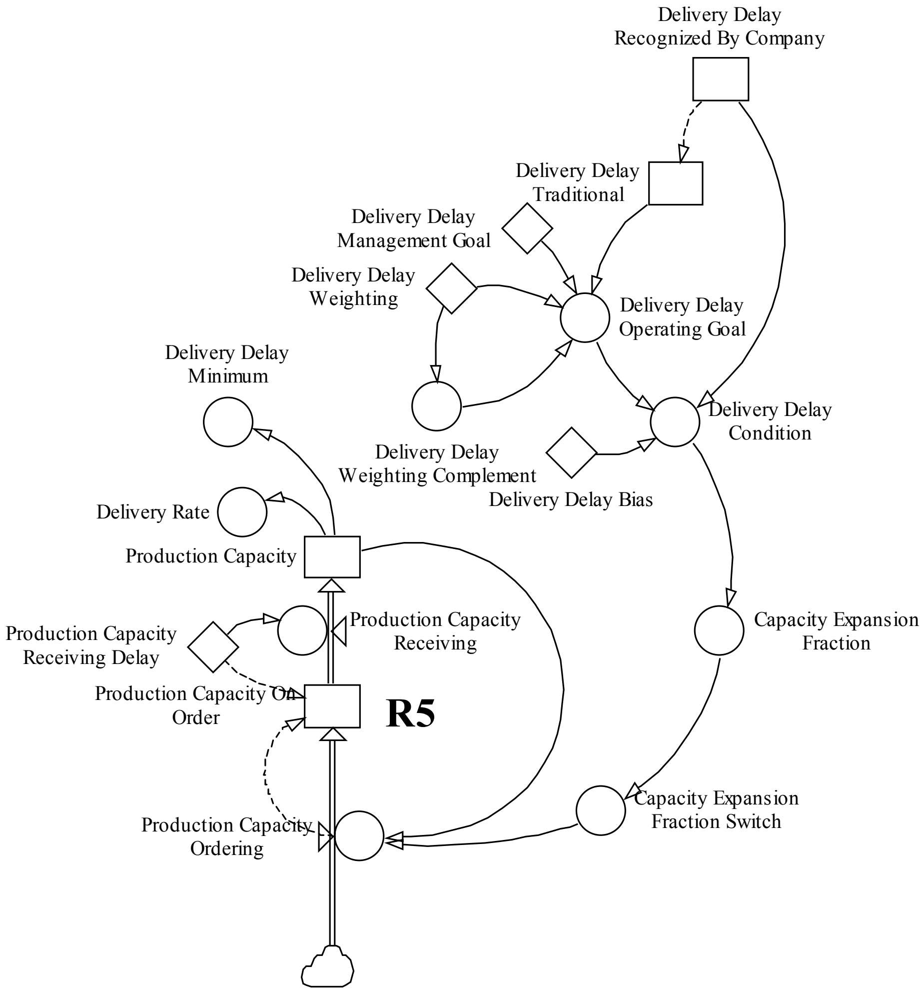


Figure 78: Capacity Expansion Sector

4.4 The Analysis of the Market Growth Model Behavior

4.4.1 Overview

Basically there are four major analysis steps:

- Step 1: List all state variables in the model and select the state variable that you want to investigate.
- Step 2: Select a meaningful set of linearly independent loops (Kampmann, 1996).
- Step 3: Plot the behavior of the chosen state variable, chop the behavior into several phases, and pick a sample time instant from each phase.
- Step 4: For each time instant, conduct an eigenvalue analysis along the following steps:
    - Step A: Compute the eigenvalues and identify the dominant one.
    - Step B: Identify the dominant links that most significantly influence the dominant eigenvalue.
    - Step C: Identify the dominant loops that most significantly influence the dominant eigenvalue.
    - Step D: Identify the dominant inputs that most significantly influence the dominant eigenvalue.

In the following, we document the results of applying this analysis to the Market Growth Model.

4.4.2 The State Variables

In Forrester's Market Growth model – in its seventh run, there are eight state variables; also, there are three states variables hidden in the third order delay associated with the *Production Capacity Receiving*. Here are the state variables:

1. *Salesmen*
2. *Backlog*
3. *Delivery Rate Average*
4. *Delivery Delay Recognized By Company*
5. *Delivery Delay Recognized By Market*
6. *Delivery Delay Traditional*
7. *Production Capacity*
8. *Production Capacity On Order*
9. *Production Capacity Receiving In Transit 1*
10. *Production Capacity Receiving In Transit 2*
11. *Production Capacity Receiving In Transit 3*

In this chapter we will focus on the behavior of the *Backlog*.

4.4.3 Linearly Independent Loops

The second step in the analysis is to identify a meaningful set of linearly independent loops. The first five loops in this set are indicated in figure 76, figure 77 and figure 78. In this model, the total number of independent loops is sixteen while the total number of loops is twenty five. The appendix contains the full list of all loops in the model, and also the set of linearly independent loops selected for this study.

4.4.4 The Behavior of the Backlog

Figure 79 portrays the behavior of the *Backlog*. By visual inspection and by utilizing **Behavior Pattern Index**[†††††] (BPI) (Saleh, 2002), it is possible to identify eleven phases through which the behavior passes during the whole simulation up until time 100.

Moreover, eleven time instants can be sampled at the center of each phase (marked by the hollow diamonds in figure 79).

Those time instants and the associated phases and phase signs are shown in table 2.

[†††††] The angle between the slope vector $\dot{\mathbf{x}}$ and the curvature vector $\ddot{\mathbf{x}}$ (Saleh, 2002).

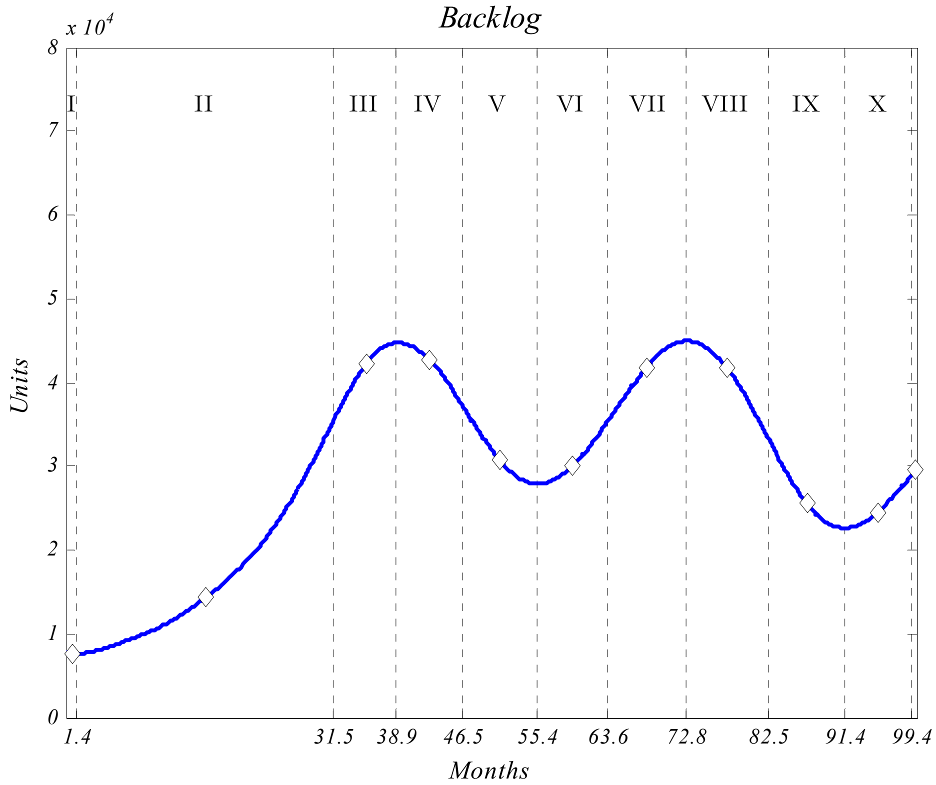


Figure 79: Behavior of Backlog

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Phase No.	I	II	III	IV	V	VI	VII	VIII	IX	X	XI
Phase Sign	-1	1	-1	1	-1	1	-1	1	-1	1	-1

Table 2: Time Instants and their Phase Signs

4.4.5 Eigenvalue Analysis at the Selected Time Instants

4.4.5.1Eigenvalues

4.4.5.1.1 Eigenvalues of the Model

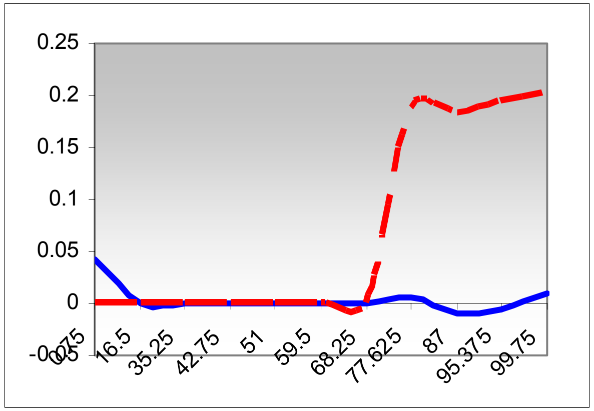
Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Eigenvalue 1	0.041717	0	0	0	0	0	0	0.0050924 +0.18757i	-0.0099171 +0.18339i	-0.0048915 +0.19492i	0.010114 +0.20154i
Eigenvalue 2	-0.29813 +0.070242i	0.03368	0.012717 +0.1936i	0.0028904	-0.021749 +0.17046i	-0.019896 +0.17244i	0.0081169 +0.19086i	-0.25918 +0.12446i	-0.27707 +0.096207i	-0.02376 +0.0091568i	-0.27122 +0.10785i
Eigenvalue 3	-0.014473	-0.082634 +0.068095i	-0.00049333 +0.0065885i	0.0051127 +0.17934i	-0.25951 +0.09252i	-0.018241 +0.010253i	-0.25618 +0.12031i	-0.0011933	-0.13406 +0.040473i	-0.13127 +0.047026i	-0.019623 +0.0062004i
Eigenvalue 4	-0.083333	-0.28276 +0.13445i	-0.2349 +0.14957i	-0.24373 +0.13536i	-0.017469 +0.0055733i	-0.15317 +0.055016i	-0.013206 +0.0035519i	-0.029613	-0.021574	-0.28045 +0.097156i	-0.14233 +0.032782i
Eigenvalue 5	-0.13927	-0.0074115	-0.14208	-0.018081	-0.15961 +0.042397i	-0.26637 +0.091917i	-0.15566	-0.12415	-0.02661	-0.43819	-0.4545
Eigenvalue 6	-0.15829	-0.16702	-0.24826	-0.13759	-0.4222	-0.42392	-0.16849	-0.18457	-0.4287	-1.0011	-0.99938
Eigenvalue 7	-0.27589	-0.36194	-0.46532	-0.21996	-1.0011	-1.0007	-0.45405	-0.45299	-1.001	-0.28045 -0.097156i	-0.14233 -0.032782i
Eigenvalue 8	-0.76705	-0.45279	-0.999	-0.45081	-0.15961 -0.042397i	-0.26637 -0.091917i	-0.99927	-0.99931	-0.13406 -0.040473i	-0.13127 -0.047026i	-0.019623 -0.0062004i
Eigenvalue 9	-0.80716	-0.95372	-0.2349 -0.14957i	-0.99923	-0.017469 -0.0055733i	-0.15317 -0.055016i	-0.013206 -0.0035519i	-0.25918 -0.12446i	-0.27707 -0.096207i	-0.02376 -0.0091568i	-0.27122 -0.10785i

Time (Months)	**0.75**	**16.5**	**35.25**	**42.75**	**51**	**59.5**	**68.25**	**77.625**	**87**	**95.375**	**99.75**
Eigenvalue 10	-0.29813 -0.070242i	-0.28276 -0.13445i	-0.00049333 -0.0065885i	-0.24373 -0.13536i	-0.25951 -0.09252i	-0.018241 -0.010253i	-0.25618 -0.12031i	0.0050924 -0.18757i	-0.0099171 -0.18339i	-0.0048915 -0.19492i	0.010114 -0.20154i
Eigenvalue 11	0	-0.082634 -0.068095i	0.012717 -0.1936i	0.0051127 -0.17934i	-0.021749 -0.17046i	-0.019896 -0.17244i	0.0081169 -0.19086i	0	0	0	0

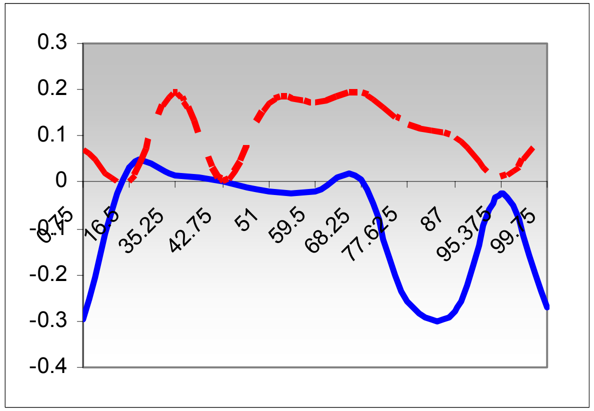
4.4.5.1.2 Rank of Eigenvalues according to Dominance‡‡‡‡‡

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Eigenvalue 1	10	8	8	7	6	5	7	2	3	2	2
Eigenvalue 2	8	1	2	3	2	2	2	9	5	11	5
Eigenvalue 3	6	4	4	2	4	11	6	3	10	9	11
Eigenvalue 4	4	6	11	5	9	9	10	11	1	4	9
Eigenvalue 5	3	2	5	11	11	4	3	10	11	5	3
Eigenvalue 6	2	7	6	9	7	7	11	4	6	7	7
Eigenvalue 7	9	10	7	10	5	6	4	5	8	3	8
Eigenvalue 8	1	11	9	6	10	3	8	7	9	8	10
Eigenvalue 9	11	9	10	8	8	8	9	8	4	10	4
Eigenvalue 10	7	5	3	4	3	10	5	1	2	1	1
Eigenvalue 11	5	3	1	1	1	1	1	6	7	6	6

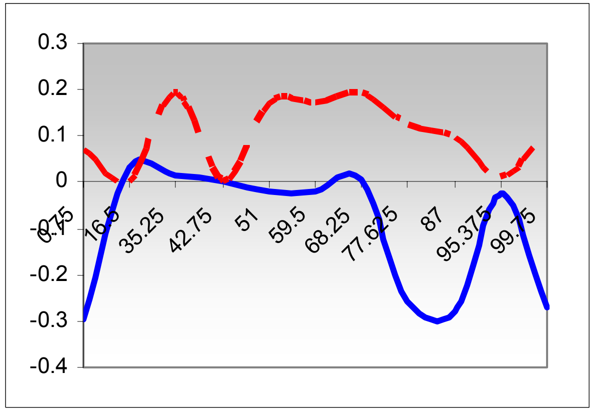
‡‡‡‡‡ Based on their percentage contribution.



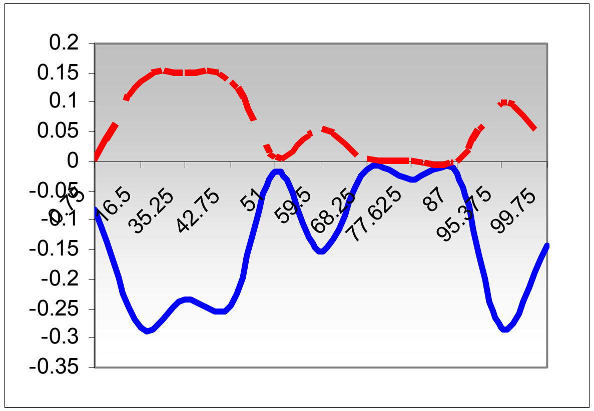
Eigenvalue 1



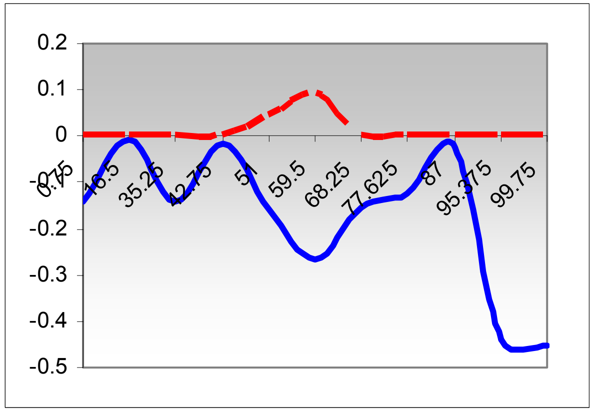
Eigenvalue 2



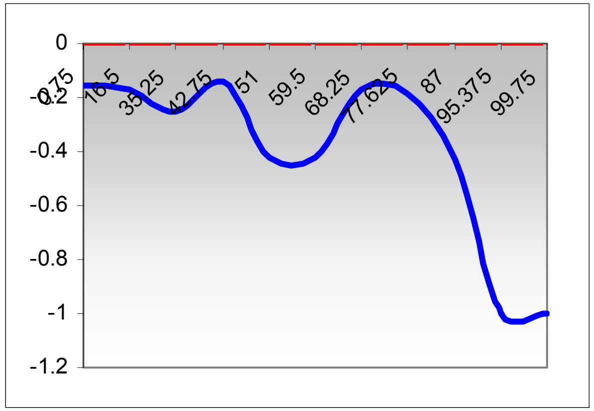
Eigenvalue 3



Eigenvalue 4



Eigenvalue 5



Eigenvalue 6

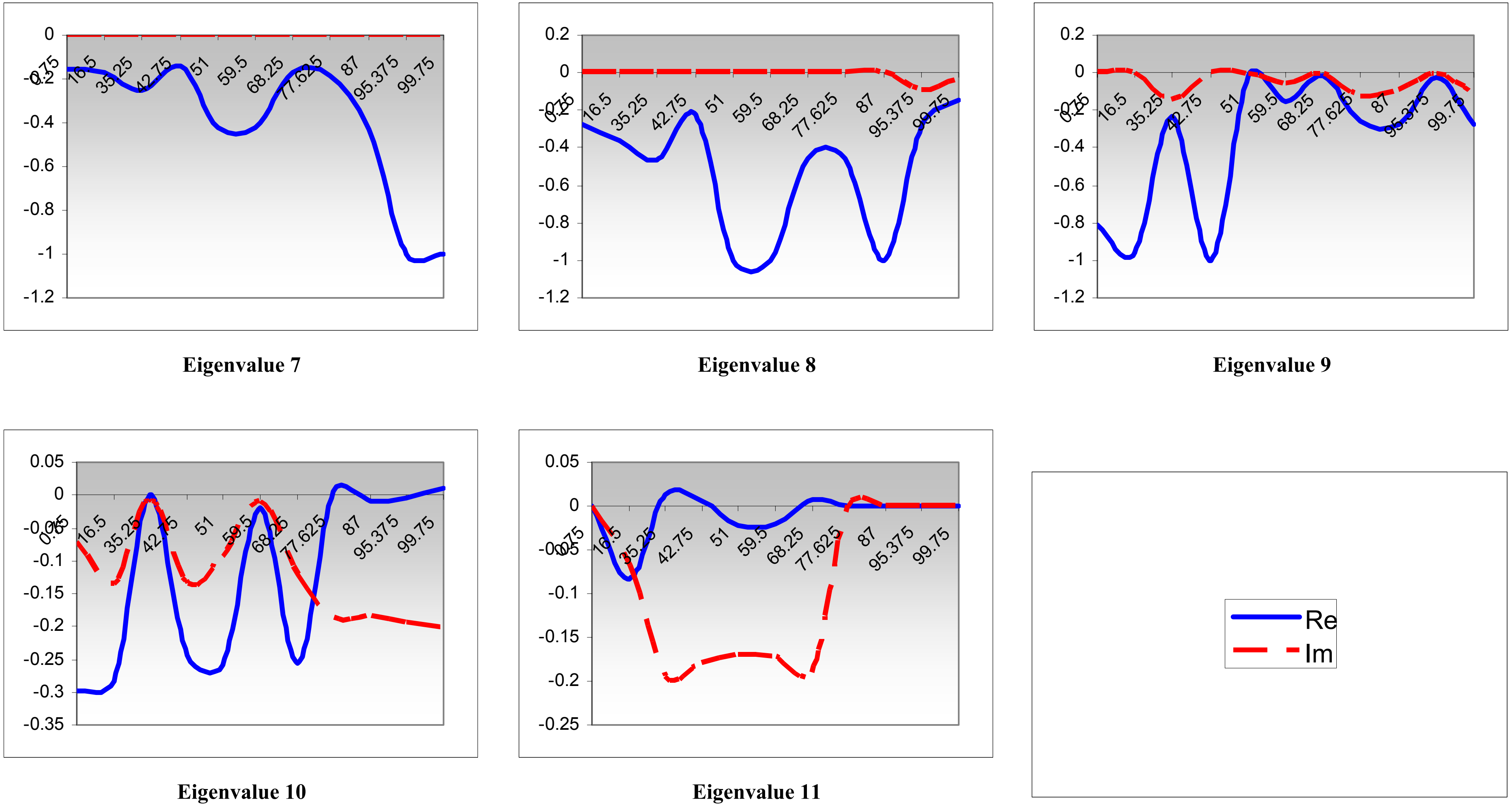


Figure 80: Plot of Eigenvalues of the Model

4.4.5.1.3 Dominant Eigenvalues

Time (Months)	**0.75**	**16.5**	**35.25**	**42.75**	**51**	**59.5**	**68.25**	**77.625**	**87**	**95.375**	**99.75**
Eigenvalue	8	2	11	11	11	11	11	10	4	10	10
Value of the Eigenvalue	-0.76705	0.03368	0.012717 -0.1936i	0.0051127 -0.17934i	-0.021749 -0.17046i	-0.019896 -0.17244i	0.0081169 -0.19086i	0.0050924 -0.18757i	-0.021574	-0.0048915 -0.19492i	0.010114 -0.20154i
Percentage Contribution §§§§§	1331.4 %	86.397 %	68.634 %	125.51 %	114.12 %	105.56 %	122.88 %	75.027 %	102.77 %	130.06 %	127.56 %
Mode of Behavior	exponential decaying	exponential growing	growing oscillatory	growing oscillatory	decaying oscillatory	decaying oscillatory	growing oscillatory	growing oscillatory	exponential decaying	decaying oscillatory	growing oscillatory
Time Constant (Months)	0.9	20.6	54.5	135.6	31.9	34.8	32.9	136.1	32.1	141.7	68.5
Time Period (Months)	x	x	32.5	35	36.9	36.4	85.4	33.5	x	32.2	31.2

§§§§§ The percentage contribution of the dominant eigenvalue might be greater than 100%, simply because the other less dominant eigenvalues have percentage contributions that might be negative. Nevertheless the total sum of all percentage contributions must equal 100%.

4.4.5.2Rank of Dominant Eigenvalue Links' Elasticity Values

4.4.5.2.1 Marginal Effect on the Real Part of the Dominant Eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 1	39	42	35	36	40	40	40	40	32	40	40
Link 2	15	16	13	13	3	3	16	13	17	9	13
Link 3	38	32	40	40	34	34	33	33	5	33	33
Link 4	14	49	2	2	2	2	2	2	34	2	2
Link 5	1	9	34	35	46	46	39	39	23	46	39
Link 6	7	6	28	25	31	31	28	25	14	32	25
Link 7	13	50	41	41	32	33	41	41	49	34	42
Link 8	10	4	26	28	29	29	27	28	12	30	28
Link 9	49	12	19	20	33	32	19	19	21	26	19
Link 10	52	52	45	45	35	38	48	48	47	39	48
Link 11	43	48	8	8	16	17	8	4	39	8	4
Link 12	33	39	39	37	21	24	30	31	6	23	32
Link 13	41	14	52	52	52	52	52	52	24	52	52

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 14	26	17	9	9	17	12	17	17	46	17	17
Link 15	34	18	43	43	41	41	44	44	48	41	44
Link 16	32	41	38	39	23	26	32	30	8	25	31
Link 17	47	47	7	7	15	16	7	8	38	7	8
Link 18	40	13	51	51	51	51	51	51	16	51	51
Link 19	18	27	23	22	19	19	22	22	41	20	21
Link 20	16	26	22	21	18	20	21	23	42	19	20
Link 21	17	28	21	23	20	18	20	21	40	18	22
Link 22	24	25	14	17	7	5	12	9	1	13	11
Link 23	36	31	36	34	37	37	38	38	30	35	38
Link 24	28	34	48	48	44	44	45	45	51	42	47
Link 25	29	33	47	47	42	43	46	47	52	43	46
Link 26	23	23	15	14	6	6	11	12	3	11	10
Link 27	27	35	46	46	43	42	47	46	50	44	45
Link 28	25	24	17	15	5	7	10	10	2	12	9
Link 29	46	46	6	6	14	15	6	7	37	6	7

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 30	48	38	50	50	50	50	50	50	22	50	49
Link 31	30	36	44	44	39	39	43	43	43	38	43
Link 32	9	3	25	27	28	28	26	27	11	29	27
Link 33	42	43	1	1	1	1	1	1	31	1	1
Link 34	3	11	33	33	48	48	37	37	26	48	37
Link 35	6	5	27	24	30	30	25	24	13	31	24
Link 36	35	30	32	32	38	36	36	36	29	37	36
Link 37	45	45	5	5	13	14	5	6	36	5	6
Link 38	31	40	37	38	22	25	31	29	7	24	30
Link 39	50	37	49	49	49	49	49	49	27	49	50
Link 40	8	2	24	26	27	27	24	26	10	28	26
Link 41	2	10	31	31	47	47	35	35	25	47	35
Link 42	44	44	4	4	12	13	4	5	35	4	5
Link 43	21	21	12	12	11	10	15	16	20	16	16
Link 44	5	8	29	29	25	22	29	32	44	27	29
Link 45	4	51	20	19	45	45	23	20	45	45	23

Time (Months)	**0.75**	**16.5**	**35.25**	**42.75**	**51**	**59.5**	**68.25**	**77.625**	**87**	**95.375**	**99.75**
Link 46	12	1	18	18	26	21	18	18	9	21	18
Link 47	11	7	3	3	8	11	3	3	15	3	3
Link 48	20	20	11	11	10	9	14	15	19	15	15
Link 49	51	15	42	42	24	23	42	42	33	22	41
Link 50	19	19	10	10	9	8	13	14	18	14	14
Link 51	22	22	16	16	4	4	9	11	4	10	12
Link 52	37	29	30	30	36	35	34	34	28	36	34

4.4.5.2.2 Marginal Effect on the Imaginary Part of the Dominant Eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 1	x	x	11	13	31	31	22	22	x	31	22
Link 2	x	x	17	22	10	14	11	14	x	13	9
Link 3	x	x	14	14	24	24	15	15	x	24	15
Link 4	x	x	45	45	52	52	45	45	x	52	45
Link 5	x	x	10	12	3	3	21	21	x	5	21
Link 6	x	x	26	21	22	19	28	28	x	22	25
Link 7	x	x	15	15	30	30	23	23	x	30	23
Link 8	x	x	29	19	21	17	26	27	x	20	28
Link 9	x	x	31	34	8	8	31	31	x	8	31
Link 10	x	x	4	4	42	42	5	5	x	42	13
Link 11	x	x	46	46	51	51	51	51	x	47	47
Link 12	x	x	42	42	35	33	40	40	x	38	39
Link 13	x	x	1	1	1	1	1	1	x	1	1
Link 14	x	x	3	3	7	7	3	3	x	4	3
Link 15	x	x	6	6	25	25	6	6	x	25	14

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 16	x	x	41	41	34	35	42	42	x	37	38
Link 17	x	x	50	50	50	50	50	50	x	51	51
Link 18	x	x	2	2	2	2	2	2	x	2	2
Link 19	x	x	36	35	36	38	36	35	x	34	34
Link 20	x	x	35	36	38	37	35	36	x	32	36
Link 21	x	x	37	37	37	36	34	34	x	33	35
Link 22	x	x	21	29	14	11	9	7	x	9	5
Link 23	x	x	12	11	28	28	20	20	x	27	20
Link 24	x	x	33	33	39	40	37	37	x	41	42
Link 25	x	x	32	32	40	39	38	39	x	39	41
Link 26	x	x	24	27	13	12	7	10	x	10	7
Link 27	x	x	34	31	41	41	39	38	x	40	40
Link 28	x	x	23	26	12	13	8	9	x	11	6
Link 29	x	x	49	49	49	49	49	49	x	50	50
Link 30	x	x	44	44	44	44	44	44	x	44	44
Link 31	x	x	39	39	32	32	33	33	x	35	33

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 32	x	x	28	18	20	16	25	26	x	19	27
Link 33	x	x	51	52	45	46	46	46	x	45	46
Link 34	x	x	9	10	5	5	19	19	x	7	19
Link 35	x	x	25	20	19	18	27	25	x	21	24
Link 36	x	x	8	9	27	27	18	18	x	28	18
Link 37	x	x	48	48	48	48	48	48	x	49	49
Link 38	x	x	40	40	33	34	41	41	x	36	37
Link 39	x	x	43	43	43	43	43	43	x	43	43
Link 40	x	x	27	17	18	15	24	24	x	18	26
Link 41	x	x	7	8	4	4	17	17	x	6	17
Link 42	x	x	47	47	47	47	47	47	x	48	48
Link 43	x	x	20	25	17	22	14	13	x	16	12
Link 44	x	x	30	30	23	23	29	29	x	23	29
Link 45	x	x	5	5	9	9	4	4	x	17	4
Link 46	x	x	38	38	6	6	32	32	x	3	32
Link 47	x	x	52	51	46	45	52	52	x	46	52

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Link 48	x	x	19	24	16	21	13	12	x	15	11
Link 49	x	x	16	16	29	29	30	30	x	29	30
Link 50	x	x	18	23	15	20	12	11	x	14	10
Link 51	x	x	22	28	11	10	10	8	x	12	8
Link 52	x	x	13	7	26	26	16	16	x	26	16

4.4.5.3Dominant Eigenvalue Inputs' Elasticity Values

4.4.5.3.1 Marginal Effect on the Real Part of the Dominant Eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Input 1	11	10	12	12	12	12	13	13	12	12	13
Input 2	2	1	5	5	5	5	5	5	2	5	4
Input 3	16	15	14	14	14	15	14	14	4	14	14
Input 4	15	16	15	15	13	13	15	15	16	13	15
Input 5	1	13	6	6	15	14	6	6	6	15	7
Input 6	3	3	3	2	2	2	2	3	13	2	2
Input 7	4	2	1	1	1	1	1	1	14	1	3
Input 8	10	9	11	11	11	11	12	12	11	11	12
Input 9	14	14	2	3	3	3	3	2	15	3	1
Input 10	12	4	4	4	6	6	4	4	5	6	5
Input 11	9	8	10	10	10	10	11	11	10	10	11
Input 12	5	12	13	13	4	4	7	7	1	4	6
Input 13	8	7	9	9	9	9	10	10	9	9	10

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Input 14	7	6	8	8	8	8	9	9	8	8	9
Input 15	13	11	16	16	16	16	16	16	3	16	16
Input 16	6	5	7	7	7	7	8	8	7	7	8

4.4.5.3.2 Marginal Effect on the Imaginary Part of the Dominant Eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Input 1	x	x	13	13	12	12	13	13	x	12	13
Input 2	x	x	14	14	5	5	14	14	x	5	14
Input 3	x	x	5	5	13	13	6	6	x	13	6
Input 4	x	x	6	6	14	14	7	7	x	14	7
Input 5	x	x	7	7	6	6	5	5	x	6	5
Input 6	x	x	2	2	2	2	2	2	x	2	2
Input 7	x	x	1	1	1	1	1	1	x	1	1
Input 8	x	x	12	12	11	11	12	12	x	11	12
Input 9	x	x	16	16	16	16	16	16	x	16	16
Input 10	x	x	4	4	4	4	4	4	x	4	4
Input 11	x	x	11	11	10	10	11	11	x	10	11
Input 12	x	x	15	15	15	15	15	15	x	15	15
Input 13	x	x	10	10	9	9	10	10	x	9	10
Input 14	x	x	9	9	8	8	9	9	x	8	9
Input 15	x	x	3	3	3	3	3	3	x	3	3

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Input 16	x	x	8	8	7	7	8	8	x	7	8

4.4.5.4Dominant Eigenvalue Independent Loops' Elasticity Values

4.4.5.4.1 Marginal Effect on the Real Part of the Dominant Eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Loop 1	3	14	8	8	5	6	8	8	13	7	8
Loop 2	2	1	4	4	4	4	4	4	3	5	4
Loop 3	1	15	10	10	14	14	13	10	10	14	13
Loop 4	15	2	3	3	6	5	3	3	5	4	3
Loop 5	11	7	7	7	7	7	6	6	1	6	6
Loop 6	16	16	11	11	8	8	14	14	11	8	14
Loop 7	14	4	16	16	16	16	16	16	6	16	16
Loop 8	10	6	9	9	10	10	9	9	12	10	9
Loop 9	13	3	15	15	15	15	15	15	4	15	15
Loop 10	8	10	14	13	12	11	12	12	15	12	11
Loop 11	7	8	13	12	11	12	10	13	16	11	10
Loop 12	6	9	12	14	13	13	11	11	14	13	12
Loop 13	4	13	2	2	2	2	1	1	8	1	1

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Loop 14	5	5	1	1	1	1	2	2	9	2	2
Loop 15	12	12	5	5	9	9	7	7	7	9	7
Loop 16	9	11	6	6	3	3	5	5	2	3	5

4.4.5.4.2 Marginal effect on the imaginary part of the dominant eigenvalue

Time (Months)	0.75	16.5	35.25	42.75	51	59.5	68.25	77.625	87	95.375	99.75
Loop 1	x	x	9	9	9	9	9	9	x	8	9
Loop 2	x	x	10	10	5	5	10	10	x	5	10
Loop 3	x	x	3	3	6	6	4	4	x	10	4
Loop 4	x	x	11	14	4	4	11	11	x	4	11
Loop 5	x	x	8	8	7	7	7	7	x	6	7
Loop 6	x	x	5	5	15	15	5	5	x	15	5
Loop 7	x	x	1	1	1	1	1	1	x	1	1
Loop 8	x	x	6	6	8	8	6	6	x	7	6
Loop 9	x	x	2	2	2	2	2	2	x	2	2
Loop 10	x	x	14	11	14	14	14	13	x	14	13
Loop 11	x	x	12	12	13	13	13	14	x	12	15
Loop 12	x	x	13	13	12	12	12	12	x	13	14
Loop 13	x	x	16	16	16	16	16	16	x	16	16
Loop 14	x	x	4	4	3	3	3	3	x	3	3
Loop 15	x	x	7	7	10	10	8	8	x	9	8

Time (Months)	**0.75**	**16.5**	**35.25**	**42.75**	**51**	**59.5**	**68.25**	**77.625**	**87**	**95.375**	**99.75**
Loop 16	x	x	15	15	11	11	15	15	x	11	12

4.5 Insight Gained

In this chapter the eigenvalue analysis approach was applied to the market growth model, focusing on the behavior of the *Backlog* level variable.

Although the simplicity of the market growth model, in a good way it reflects the firm exerted efforts to market its products. The management of such a firm need knowledge about key elements or leverage points that could help them to strengthen desirable behavior or weaken undesirable behavior.

Concerning the benefits of using the eigenvalue analysis and its results to help mangers and decision takers, some concepts should be clear first; if the aim is changing the difference between the maximum and minimum limits of the *backlog* –as it is the level to study in this context–, the management of the firm should take into consideration the variables that have larger marginal effect on the real part of the dominant eigenvalue. While if the management is interested in changing the periodic time of the cycles in the *backlog*, the variables that have larger values in its marginal effect on the imaginary part of the dominant eigenvalue are the variables to get the focus.

When checking the marginal effect on either real or imaginary parts of the dominant eigenvalue, three kinds of results could be checked; these dominant eigenvalue elasticity values of model's linearly independent loops, links or inputs. In this context, the linearly independent loops 14 and 13 share the first position when talking about the marginal effect on the real part of the dominant eigenvalue and this is for most of the simulation time. While linearly independent loops 7, 9 and 14 have the largest value of the marginal effect on the imaginary part of the dominant eigenvalue.

According to the results of the eigenvalue analysis of the model's links; link 47 (*Orders Booked* → *Backlog*), link 4 (*Backlog* → *Delivery Delay Indicated*) and link 33 (*Delivery Delay Indicated* → *Delivery Delay Recognized By Company Adjustment*) in order, are the links that have the largest marginal effect on the real part of the dominant eigenvalue most of the simulation time. While link 13 (*Delivery Delay Recognized By Company* → *Delivery Delay Recognized By Company Adjustment*), link 18 (*Delivery Delay Recognized By Market* → *Delivery Delay Recognized By Market Adjustment*) and link 14 (*Delivery Delay Recognized By Company* → *Delivery Delay Condition*) in order, are the links that have the largest marginal effect on the imaginary part of the dominant eigenvalue most of the simulation time.

By checking figure 81 and figure 82; it could be easily identified that both links 33 and 4 are elements of both linearly independent loops 13 and 14, and link 47 is an element of linearly independent loop 13. While links 13, 18 and 14 are elements of linearly independent loops 7, 9 and 14 respectively.

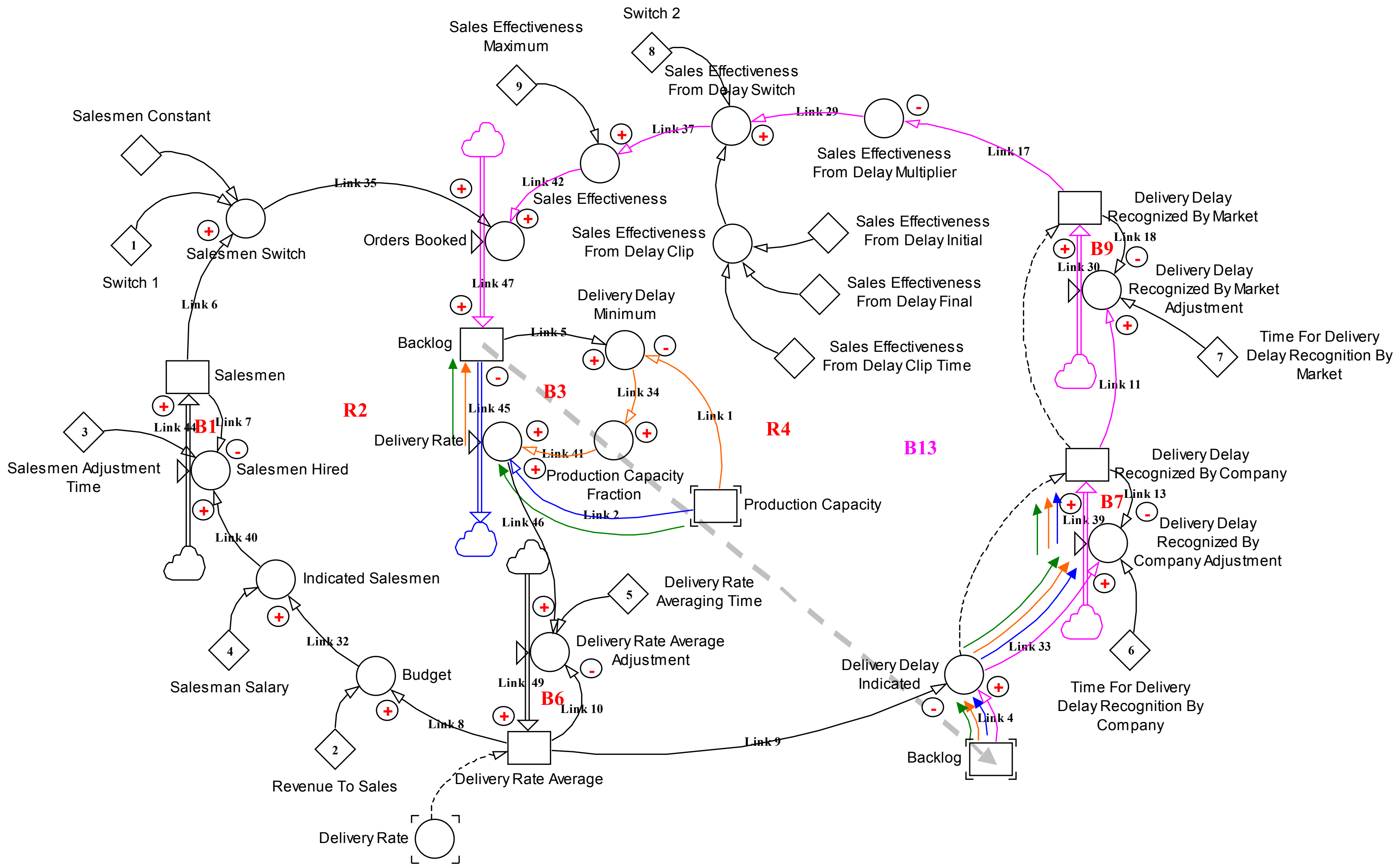


Figure 81: Market Growth Model Loops 1, 2, 3, 4, 6, 7, 9 and 13

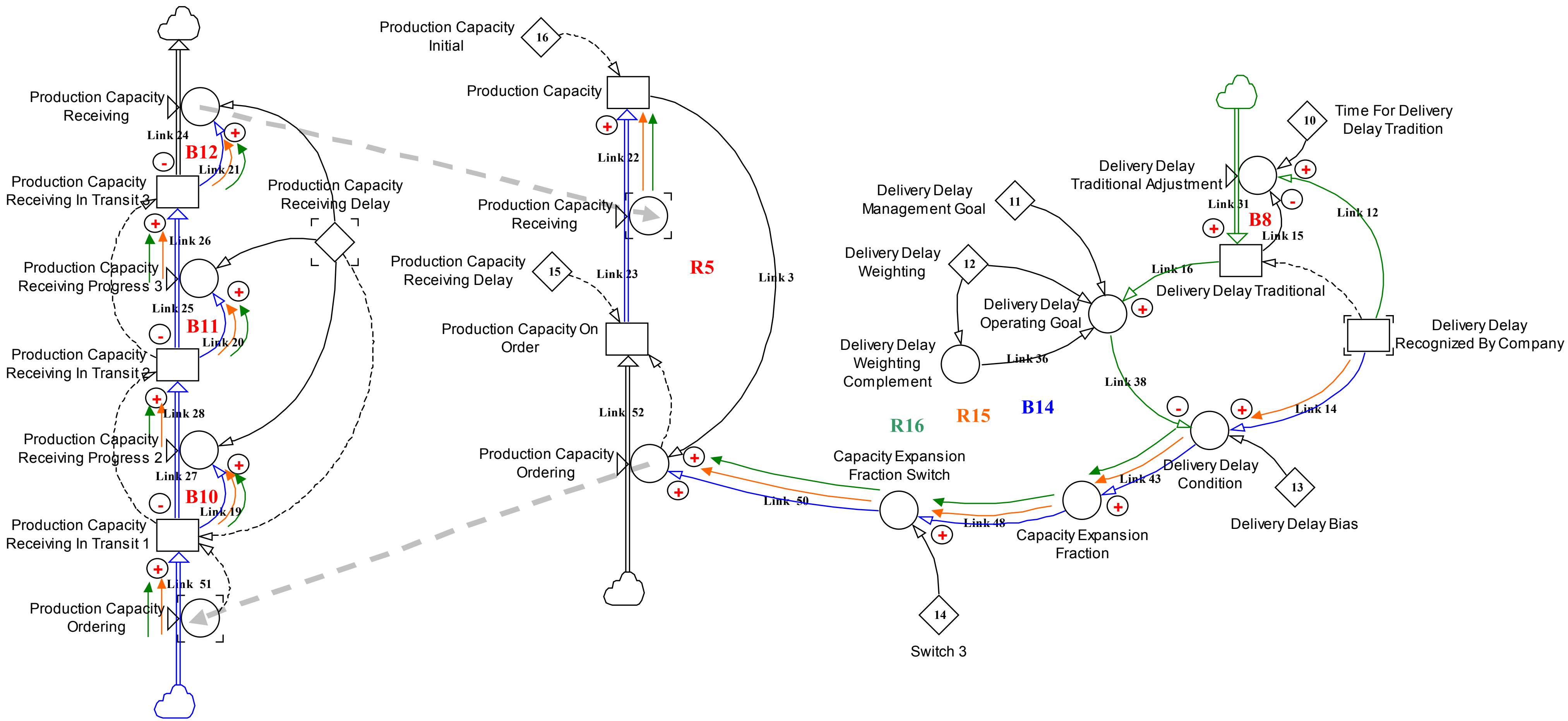


Figure 82: Market Growth Model Loops 5, 8, 10, 11, 12, 14, 15 and 16

Testing management policies to decide the best policy is valuable, but not enough. Somehow managers and decision takers would appreciate suggesting new policies. In other words the system dynamics paradigm won't fulfill management needs by just simulating the policies designed by this management. It should be in the course of designing policies based on their very well-known needs, for example; they might ask for damping the oscillations presented in the behavior of the *Backlog* or increasing the period of that oscillations or how to increase or decrease time constant of the envelope of that oscillations.

As stated before, inputs are the controllable objects in a system dynamics model. So that, by reviewing the table of inputs elasticity values associated with the dominant eigenvalue, it would be easily noted that *Time For Delivery Delay Recognition By Market* takes rank 1 most of the time for both marginal effect on both the real and the imaginary parts of the dominant eigenvalue; which indicates that to control the behavior of the *Backlog*, *Time For Delivery Delay Recognition By Market* should be changed according to its elasticity value associated with the dominant eigenvalue. *Time For Delivery Delay Recognition By Market* has positive elasticity value nearly all the time, which means that an increase in its gain consequently would increase both real and imaginary parts of the dominant eigenvalue and vice versa, in other words they are directly proportional to each other.

The marginal effect of *Time For Delivery Delay Recognition By Market* on the real part of the dominant eigenvalue on average equals 0.14, while its marginal effect on the imaginary part of the dominant eigenvalue on average equals 0.27. Changing its value by 10%, leads to an increase around 1.4% in the real part of the dominant eigenvalue shown in figure 83 (the dashed line), and an increase around 2.7% in the imaginary part of the dominant eigenvalue shown in figure 84 (the dashed line).

Time (Months)	**Dominant Eigenvalue**	**"*Time For Delivery Delay Recognition By Market*" Elasticity Value Associated with the Dominant Eigenvalue**		
		Total Effect	**Marginal Effect on Real Part of the Dominant Eigenvalue**	**Marginal Effect on Imaginary Part of the Dominant Eigenvalue**
0.75	-0.76705	0.40479	0.40479	x
16.5	0.03368	0.025932	0.025932	x
35.25	0.012717 - 0.1936i	-0.23866 +0.1226i	0.10669	0.24619
42.75	0.0051127 - 0.17934i	-0.24021 +0.13257i	0.12567	0.24389
51	-0.021749 - 0.17046i	-0.28977 +0.1661i	0.20144	0.26642
59.5	-0.019896 - 0.17244i	-0.29156 +0.17271i	0.20499	0.26984
68.25	0.0081169 - 0.19086i	-0.26772 +0.13245i	0.12096	0.2731
77.63	0.0050924 - 0.18757i	-0.27304 +0.12885i	0.12139	0.27644
87	-0.021574	-0.049707	-0.049707	x
95.38	-0.0048915 - 0.19492i	-0.31705 +0.13292i	0.14083	0.31361
99.75	0.010114 - 0.20154i	-0.2947 +0.12577i	0.11084	0.30063

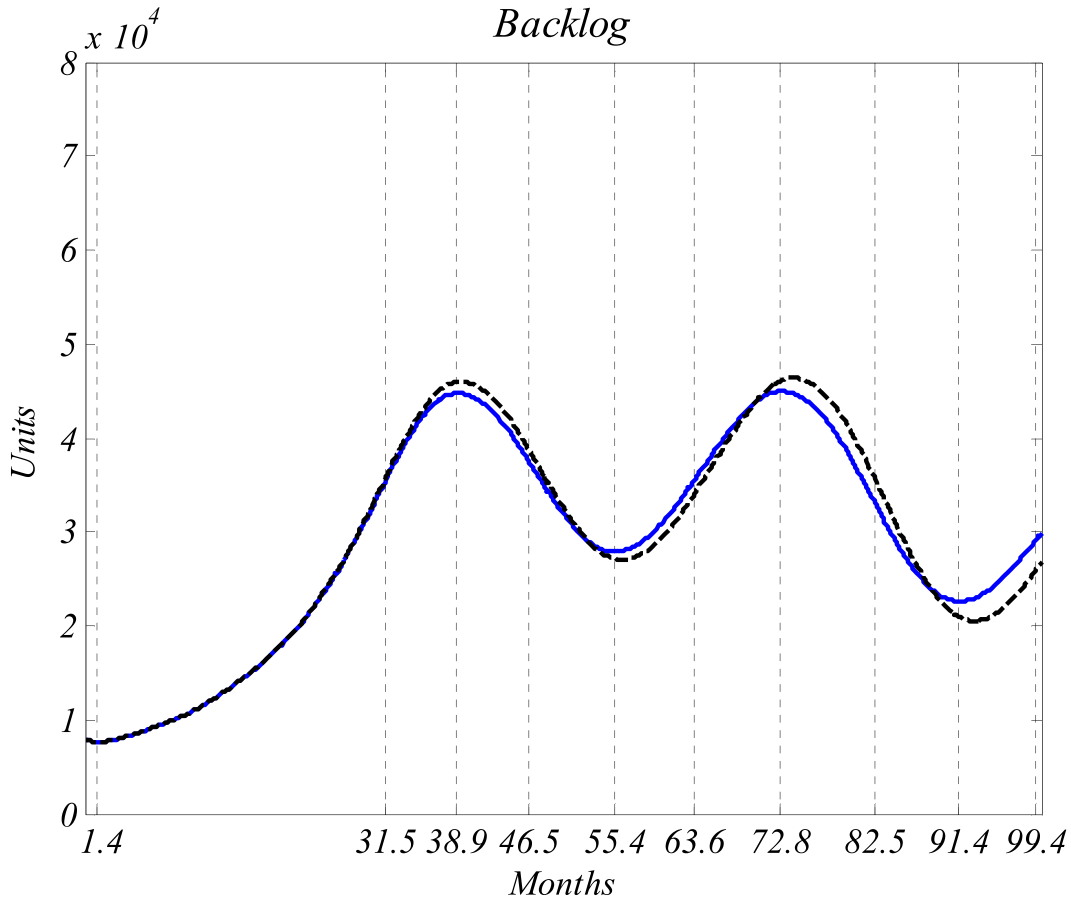


Figure 83: Behavior of Backlog after 10% Increase in "*Time For Delivery Delay Recognition By Market*"

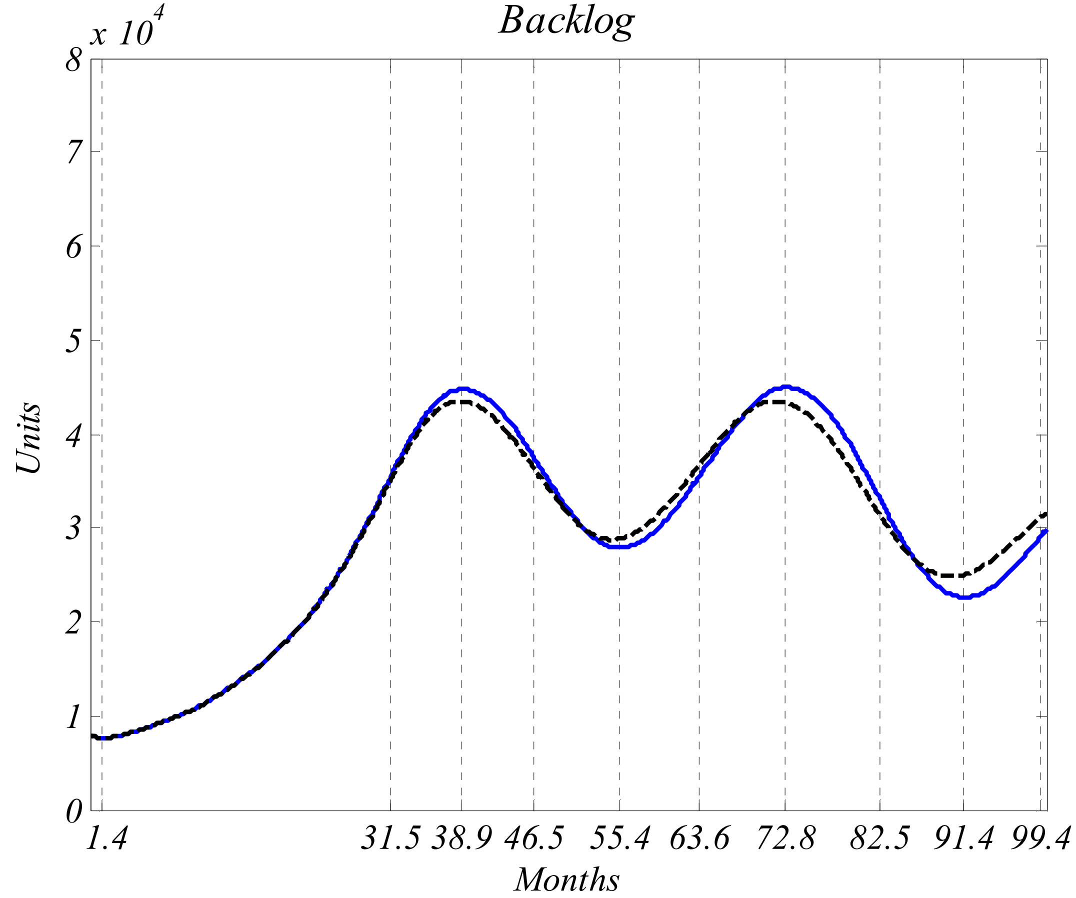


Figure 84: Behavior of Backlog after 10% Decrease in "*Time For Delivery Delay Recognition By Market*"

Chapter 5

Conclusion and Future Extensions

5.1 Conclusion

The purpose of this thesis is divided into three main targets:

1. Standardizing the concepts and mathematical symbolic expressions in the field of system dynamics models analysis, and in the same time comply with the control theory concepts and mathematical symbols
2. Design and implement a computer package to perform the eigenvalue analysis process on a system dynamics model
3. Apply eigenvalue analysis on the Market Growth Model of Jay W. Forrester

In the light of the purpose stated above, in the first chapter; a brief introduction to the idea has been given, as well as reviewing some related literature.

In the second chapter; a mathematical and theoretical background has been established taking the standardization issue into consideration.

In the third chapter; a full design and implementation of the Analysis package on Matlab was introduced and fully explained.

In the fourth chapter; an interesting application of the eigenvalue analysis to the Market Growth model was introduced as a case study, giving a brief on the model and its important technical aspects like its states, links and loops, moreover showing the analysis steps as well as the results and experiments to test these results.

In the fifth chapter; a conclusion to the whole work is given as well as mentioning the aimed at extensions and future work.

5.2 Future Extensions

5.2.1 Theoretical Extensions

Eigenvalue analysis was applied successfully to linear as well as non-linear system dynamics models (after applying linearization) under the conditions of that models being time-invariant and deterministic; further investigation should be devoted to the application of eigenvalue analysis to time-variant models as well as stochastic models, because models of these kinds cover a wide range of system dynamics models; especially in the filed of economics where models deal with exogenous variables in the form of time series, also where relations among variables have a probabilistic fashion.

5.2.2 Implementation Extensions

In this version of the Analysis package; models with time delays (models containing variables delayed in time) are not dealt with directly, i.e. the user has to replace these variables delayed in time with their proper replacements using additional stocks and flows; so that some further implementation is needed to automate this replacement process, and keep the user away.

Also this version is based on another commercial package, so that it is badly needed to implement it as a stand alone program, and this could be done by one of two ways:

1. Leave the Matlab code and re-implement the whole package into any other programming language and then compile it to a stand alone program.

2. Let Matlab to convert the Analysis package code into C language and compile it directly to a stand alone program.

In fact, the second option is not possible with the current version of the Analysis package; because this version depends highly on Matlab symbolic toolbox, at the same time, code depending on this toolbox can not be converted into a stand alone program unless a replacement to the symbolic toolbox on Matlab is used.

5.2.3 More Functions Extensions

In this version of the Analysis package; only graph (table) function and if-then-else relation are implemented, this is of course in addition to the ordinary mathematical operations and functions (addition, multiplication, exponential function … etc.) that did not need implementation because Matlab already has them.

More functions need to be implemented in order to give the package the capability to analyze a wider range of system dynamics models.

References

Bahar, M. and J. Jantzen (1995). Digraph toolbox.

Belikov, B. S. (1986). General methods for solving physics problems. Moscow, Mir publishers.

Beltrami, E. (1998). Mathematics for Dynamic Modeling, Academic Press.

Bowman, C. F. (1994). Algorithms and Data Structures an Approach in C. New York, Oxford University Press Inc.

Deif, A. S. (1982). Advanced Matrix Theory for Scientists and Engineers, Abacus Press.

Diestel, R. (2000). Graph Theory. New York, Springer-Verlag.

Dorf, R. C. and R. H. Bishop (2000). Modern Control Systems, Prentice Hall.

Edminister, J. (1972). Schaum's Outline of Electric Circuits, McGraw-Hill.

Forrester, J. W. (1968). Market Growth as Influenced by Capital Investment. Industrial Management Rev. (MIT). **9:** 83-105.

Forrester, J. W. (1975). Market Growth as Influenced by Capital Investment. Collected Papers of Jay W. Forrester. Cambridge MA, Productivity Press.

Forrester, J. W. (1978). Market Growth as Influenced by Capital Investment. Managerial Applications of System Dynamics. E. B. Roberts. Cambridge MA, Productivity Press.

Forrester, N. B. (1982). A Dynamic Synthesis of Basic Macroeconomic Theory: Implications for Stabilization Policy Analysis, M. I. T.

Forrester, N. B. (1983). Eigenvalue Analysis of Dominant Feedback Loops. Intl. System Dynamics Conf., Chestnut Hill, MA.

Friedland, B. (1987). Control System Design, McGraw-Hill Book Company.

Gopal, M. (1993). Modern Control Theory, Wiley Eastern Limited.

Gordon, G. (1989). System Simulation. New Delhi, Prentice-Hall of India.

Grantham, W. J. and T. L. Vincent (1993). Modern Control Systems Analysis and Design, John Wiley & Sons, Inc.

Hanselman, D. C. and B. R. Littlefield (1997). Mastering MATLAB 5: A Comprehensive Tutorial and Reference, Prentice Hall.

Kampmann, C. E. (1996). Feedback Loop Gains and System Behavior. 1996 International System Dynamics Conference, Cambridge, Massachusetts, System Dynamics Society.

Kendall, K. E. and Kendall, J. E. (2002). Systems Analysis and Design. Upper Saddle River, NJ, Prentice-Hall Inc.

Kheir, N. A. (1996). Systems Modeling and Computer Simulation. New York, Marcel Dekker Inc.

Kreyszig, E. (1993). Advanced Engineering Mathematics. New York, John Wiley & Sons, Inc.

Kuo, B. C. (1995). Automatic Control Systems. Englewood Cliffs, NJ, Prentice-Hall, Inc.

Nise, N. S. (1994). Control Systems Engineering, John Wiley & Sons, Inc.

Ogata, K. (1997). Modern Control Engineering. Upper Saddle River, NJ, Prentice-Hall Inc.

Palm, W. J. (2000). Introduction to Matlab 6 for Engineers, McGraw-Hill Science/Engineering/Math.

Richardson, G. P. and A. L. Pugh, III (1981). Introduction to System Dynamics Modeling with DYNAMO. Cambridge MA, Productivity Press.

Saleh, M. (1998). An Object Oriented Approach to Automate System Dynamics Model Optimization, Department of Information Science, University of Bergen, Norway.

Saleh, M. (2002). The Characterization of Model Behavior and its Casual Foundation, Department of Information Science, University of Bergen, Norway.

Saleh, M. and P. I. Davidsen (2000). An Eigenvalue Approach To Feedback Loop Dominance Analysis In Non-Linear Dynamic Models. 18th International Conference of the System Dynamics Society, Bergen, Norway, System Dynamics Society.

Saleh, M. and P. I. Davidsen (2001). The Origins of Business Cycles. The 19th International Conference of the System Dynamics Society, Atlanta, Georgia, System Dynamics Society.

Sterman, J. D. (2000). Business Dynamics : Systems Thinking and Modeling for a Complex World. Boston, Irwin/McGraw-Hill.

The Mathworks (2002). Using Matlab Version 6, The Mathworks, Inc.

The Mathworks (2002). Using Simulink Version 5, The Mathworks, Inc.

Weisstein, E. W. (1999). Concise Encyclopedia of Mathematics CD-ROM.

White, J. (2004). System Dynamics (22.554 and 24.509).

Appendix A

Market Growth Model Equations

A.1 Model Equations

Market Growth Model equation details conform to the Powersim environment:

□ Backlog
 ◇ 8000
 ⇦ -dt*Delivery Rate
 ⇨ +dt*Orders Booked
 🌡 Unit
□ Delivery Delay Recognized By Company
 ◇ Delivery Delay Indicated
 ⇨ +dt*Delivery Delay Recognized By Company Adjustment
 🌡 Month
□ Delivery Delay Recognized By Market
 ◇ Delivery Delay Recognized By Company
 ⇨ +dt*Delivery Delay Recognized By Market Adjustment
 🌡 Month
□ Delivery Delay Traditional
 ◇ Delivery Delay Recognized By Company
 ⇨ +dt*Delivery Delay Traditional Adjustment
 🌡 Month
□ Delivery Rate Average
 ◇ Delivery Rate
 ⇨ +dt*Delivery Rate Average Adjustment
 🌡 Unit/Month
□ Production Capacity
 ◇ Production Capacity Initial
 ⇨ +dt*Production Capacity Receiving
 🌡 Unit/Month
□ Production Capacity On Order
 ◇ Production Capacity Ordering*Production Capacity Receiving Delay
 ⇦ -dt*Production Capacity Receiving
 ⇨ +dt*Production Capacity Ordering
 🌡 Unit/Month
□ Production Capacity Receiving In Transit 1
 ◇ Production Capacity Ordering*(ProductionCapacityReceivingDelay/3)
 ⇦ -dt*Production Capacity Receiving Progress 2
 ⇨ +dt*Production Capacity Ordering
 🌡 Unit/Month
□ Production Capacity Receiving In Transit 2

- ◇ Production Capacity Receiving In Transit 1
  - ⇦ -dt*Production Capacity Receiving Progress 3
  - ⇨ +dt*Production Capacity Receiving Progress 2
  - 🌡 Unit/Month
- □ Production Capacity Receiving In Transit 3
  - ◇ Production Capacity Receiving In Transit 2
  - ⇦ -dt*Production Capacity Receiving
  - ⇨ +dt*Production Capacity Receiving Progress 3
  - 🌡 Unit/Month
- □ Salesmen
  - ◇ 10
  - ⇨ +dt*Salesmen Hired
  - 🌡 Man
- ⇨ Delivery Delay Recognized By Company Adjustment
  - (Delivery Delay Indicated-Delivery Delay Recognized By Company)
  - ○ /Time For Delivery Delay Recognition By Company
  - 🌡 Month/Month
- ⇨ Delivery Delay Recognized By Market Adjustment
  - (Delivery Delay Recognized By Company-Delivery Delay Recognized By Market)
  - ○ /Time For Delivery Delay Recognition By Market
  - 🌡 Month/Month
- ⇨ Delivery Delay Traditional Adjustment
  - (Delivery Delay Recognized By Company-Delivery Delay Traditional)
  - ○ /Time For Delivery Delay Tradition
  - 🌡 Month/Month
- ⇨ Delivery Rate
  - ○ Production Capacity*Production Capacity Fraction
  - 🌡 Unit/Month
- ⇨ Delivery Rate Average Adjustment
  - ○ (Delivery Rate-Delivery Rate Average)/Delivery Rate Averaging Time
  - 🌡 (Unit/Month)/Month
- ⇨ Orders Booked
  - ○ Sales Effectiveness*Salesmen Switch
  - 🌡 Unit/Month
- ⇨ Production Capacity Ordering
  - ○ Production Capacity*Capacity Expansion Fraction Switch
  - 🌡 (Unit/Month)/Month
- ⇨ Production Capacity Receiving
  - Production Capacity Receiving In Transit 3
  - ○ /(ProductionCapacityReceivingDelay/3)

🌡 (Unit/Month)/Month

⇨ Production Capacity Receiving Progress 2

Production Capacity Receiving In Transit 1

○ /(ProductionCapacityReceivingDelay/3)

🌡 (Unit/Month)/Month

⇨ Production Capacity Receiving Progress 3

Production Capacity Receiving In Transit 2

○ /(ProductionCapacityReceivingDelay/3)

🌡 (Unit/Month)/Month

⇨ Salesmen Hired

○ (Indicated Salesmen-Salesmen)/Salesmen Adjustment Time

🌡 Man/Month

○ Budget

○ Delivery Rate Average*Revenue To Sales

🌡 Dollar

○ Capacity Expansion Fraction

GRAPH(DeliveryDelayCondition,0,0.5,[-0.07,-0.02,0,0.02,0.07,0.15"Min:-

○ 0.07;Max:0.15"])

🌡 1/Month

○ Capacity Expansion Fraction Switch

○ IF(Switch3=0,0,CapacityExpansionFraction)

🌡 1/Month

○ Delivery Delay Condition

(Delivery Delay Recognized By Company/Delivery Delay Operating Goal)

○ -Delivery Delay Bias

🌡 Dimensionless

○ Delivery Delay Indicated

○ Backlog/Delivery Rate Average

🌡 Month

○ Delivery Delay Minimum

○ Backlog/Production Capacity

🌡 Month

○ Delivery Delay Operating Goal

DeliveryDelayTraditional

*DeliveryDelayWeighting+DeliveryDelayManagementGoal

○ *DeliveryDelayWeightingComplement

🌡 Month

○ Delivery Delay Weighting Complement

○ 1-DeliveryDelayWeighting

🌡 Dimensionless

○ Indicated Salesmen

- ○ Budget/Salesman Salary
- 🌡 Man

○ Production Capacity Fraction
- GRAPH(DeliveryDelayMinimum,0,0.5,[0,0.25,0.5,0.67,0.8,0.87,0.93,0.95,0.9
- ○ 7,0.98,1"Min:0;Max:1"])
- 🌡 Dimensionless

○ Sales Effectiveness
- ○ Sales Effectiveness From Delay Switch*Sales Effectiveness Maximum
- 🌡 (Unit/Month)/Man

○ Sales Effectiveness From Delay Clip
- IF(TIME>=SalesEffectivenessFromDelayClipTime,SalesEffectivenessFromD
- ○ elayFinal,SalesEffectivenessFromDelayInitial)
- 🌡 Dimensionless

○ Sales Effectiveness From Delay Multiplier
- GRAPH(DeliveryDelayRecognizedByMarket,0,1,[1,0.97,0.87,0.73,0.53,0.38,
- ○ 0.25,0.15,0.08,0.03,0.02"Min:0;Max:1"])
- 🌡 Dimensionless

○ Sales Effectiveness From Delay Switch
- IF(Switch2=0,SalesEffectivenessFromDelayClip,SalesEffectivenessFromDela
- ○ yMultiplier)
- 🌡 Dimensionless

○ Salesmen Switch
- ○ IF(Switch1=0,SalesmenConstant,Salesmen)
- 🌡 Man

◇ Delivery Delay Bias
- ◇ 0.3
- 🌡 Dimensionless

◇ Delivery Delay Management Goal
- ◇ 2
- 🌡 Month

◇ Delivery Delay Weighting
- ◇ 1
- 🌡 Dimensionless

◇ Delivery Rate Averaging Time
- ◇ 1
- 🌡 Month

◇ Production Capacity Initial
- ◇ 12000
- 🌡 Unit/Month

◇ Production Capacity Receiving Delay
- ◇ 12

- Month

◇ Revenue To Sales
- ◇ 12
- Dollar/Unit

◇ Sales Effectiveness From Delay Clip Time
- ◇ 36
- Month

◇ Sales Effectiveness From Delay Final
- ◇ 1
- Dimensionless

◇ Sales Effectiveness From Delay Initial
- ◇ 1
- Dimensionless

◇ Sales Effectiveness Maximum
- ◇ 400
- (Unit/Month)/Man

◇ Salesman Salary
- ◇ 2000
- Dollar/Man

◇ Salesmen Adjustment Time
- ◇ 20
- Month

◇ Salesmen Constant
- ◇ 60
- Man

◇ Switch1
- ◇ 1
- Dimensionless

◇ Switch2
- ◇ 1
- Dimensionless

◇ Switch3
- ◇ 1
- Dimensionless

◇ Time For Delivery Delay Recognition By Company
- ◇ 4
- Month

◇ Time For Delivery Delay Recognition By Market
- ◇ 6
- Month

◇ Time For Delivery Delay Tradition

◇ 12

🌡 Month

🕒 start 0

🕒 stop 100

🕒 dt 0.5

🕒 method Euler(fixed step)

Appendix B

Market Growth Model Links, Inputs and Loops

B.1 Market Growth Model Links

	Link Description
Link 1	*ProductionCapacity → DeliveryDelayMinimum*
Link 2	*ProductionCapacity → DeliveryRate*
Link 3	*ProductionCapacity → ProductionCapacityOrdering*
Link 4	*Backlog → DeliveryDelayIndicated*
Link 5	*Backlog → DeliveryDelayMinimum*
Link 6	*Salesmen → SalesmenSwitch*
Link 7	*Salesmen → SalesmenHired*
Link 8	*DeliveryRateAverage → Budget*
Link 9	*DeliveryRateAverage → DeliveryDelayIndicated*
Link 10	*DeliveryRateAverage → DeliveryRateAverageAdjustment*
Link 11	*DeliveryDelayRecognizedByCompany → DeliveryDelayRecognizedByMarketAdjustment*
Link 12	*DeliveryDelayRecognizedByCompany → DeliveryDelayTraditionalAdjustment*
Link 13	*DeliveryDelayRecognizedByCompany → DeliveryDelayRecognizedByCompanyAdjustment*
Link 14	*DeliveryDelayRecognizedByCompany → DeliveryDelayCondition*
Link 15	*DeliveryDelayTraditional → DeliveryDelayTraditionalAdjustment*
Link 16	*DeliveryDelayTraditional → DeliveryDelayOperatingGoal*
Link 17	*DeliveryDelayRecognizedByMarket → SalesEffectivenessFromDelayMultiplier*
Link 18	*DeliveryDelayRecognizedByMarket → DeliveryDelayRecognizedByMarketAdjustment*
Link 19	*ProductionCapacityReceivingInTransit 1 → ProductionCapacityReceivingProgress 2*
Link 20	*ProductionCapacityReceivingInTransit 2 → ProductionCapacityReceivingProgress 3*
Link 21	*ProductionCapacityReceivingInTransit 3 → ProductionCapacityReceiving*
Link 22	*ProductionCapacityReceiving → ProductionCapacity*
Link 23	*ProductionCapacityReceiving → ProductionCapacityOnOrder*
Link 24	*ProductionCapacityReceiving → ProductionCapacityReceivingInTransit 3*
Link 25	*ProductionCapacityReceivingProgress 3 → ProductionCapacityReceivingInTransit 2*
Link 26	*ProductionCapacityReceivingProgress 3 → ProductionCapacityReceivingInTransit 3*
Link 27	*ProductionCapacityReceivingProgress 2 → ProductionCapacityReceivingInTransit 1*
Link 28	*ProductionCapacityReceivingProgress 2 → ProductionCapacityReceivingInTransit 2*

	Link Description
Link 29	*SalesEffectivenessFromDelayMultiplier* → *SalesEffectivenessFromDelaySwitch*
Link 30	*DeliveryDelayRecognizedByMarketAdjustment* → *DeliveryDelayRecognizedByMarket*
Link 31	*DeliveryDelayTraditionalAdjustment* → *DeliveryDelayTraditional*
Link 32	*Budget* → *IndicatedSalesmen*
Link 33	*DeliveryDelayIndicated* → *DeliveryDelayRecognizedByCompanyAdjustment*
Link 34	*DeliveryDelayMinimum* → *ProductionCapacityFraction*
Link 35	*SalesmenSwitch* → *OrdersBooked*
Link 36	*DeliveryDelayWeightingComplement* → *DeliveryDelayOperatingGoal*
Link 37	*SalesEffectivenessFromDelaySwitch* → *SalesEffectiveness*
Link 38	*DeliveryDelayOperatingGoal* → *DeliveryDelayCondition*
Link 39	*DeliveryDelayRecognizedByCompanyAdjustment* → *DeliveryDelayRecognizedByCompany*
Link 40	*IndicatedSalesmen* → *SalesmenHired*
Link 41	*ProductionCapacityFraction* → *DeliveryRate*
Link 42	*SalesEffectiveness* → *OrdersBooked*
Link 43	*DeliveryDelayCondition* → *CapacityExpansionFraction*
Link 44	*SalesmenHired* → *Salesmen*
Link 45	*DeliveryRate* → *Backlog*
Link 46	*DeliveryRate* → *DeliveryRateAverageAdjustment*
Link 47	*OrdersBooked* → *Backlog*
Link 48	*CapacityExpansionFraction* → *CapacityExpansionFractionSwitch*
Link 49	*DeliveryRateAverageAdjustment* → *DeliveryRateAverage*
Link 50	*CapacityExpansionFractionSwitch* → *ProductionCapacityOrdering*
Link 51	*ProductionCapacityOrdering* → *ProductionCapacityReceivingInTransit 1*
Link 52	*ProductionCapacityOrdering* → *ProductionCapacityOnOrder*

B.2 Market Growth Model Inputs

	Input Name
Input 1	*Switch1*
Input 2	*RevenueToSales*
Input 3	*SalesmenAdjustmentTime*
Input 4	*SalesmanSalary*
Input 5	*DeliveryRateAveragingTime*
Input 6	*TimeForDeliveryDelayRecognitionByCompany*
Input 7	*TimeForDeliveryDelayRecognitionByMarket*
Input 8	*Switch2*
Input 9	*SalesEffectivenessMaximum*
Input 10	*TimeForDeliveryDelayTradition*
Input 11	*DeliveryDelayManagementGoal*
Input 12	*DeliveryDelayWeighting*
Input 13	*DeliveryDelayBias*
Input 14	*Switch3*
Input 15	*ProductionCapacityReceivingDelay*
Input 16	*ProductionCapacityInitial*

B.3 Market Growth Model Loops

	Loop Description
Loop 1	*Salesmen → Salesmen Hired*
Loop 2	*Delivery Rate Average → Delivery Rate Average Adjustment*
Loop 3	*Delivery Delay Recognized By Company → Delivery Delay Recognized By Company Adjustment*
Loop 4	*Delivery Delay Traditional → Delivery Delay Traditional Adjustment*
Loop 5	*Delivery Delay Recognized By Market → Delivery Delay Recognized By Market Adjustment*
Loop 6	*Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2*
Loop 7	*Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3*
Loop 8	*Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 9	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate*
Loop 10	*Production Capacity → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 11	*Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Recognized By Market Adjustment → Delivery Delay Recognized By Market → Sales Effectiveness From Delay Multiplier → Sales Effectiveness From Delay Switch → Sales Effectiveness → Orders Booked*
Loop 12	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked*
Loop 13	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Recognized By Market Adjustment → Delivery Delay Recognized By Market → Sales Effectiveness From Delay Multiplier → Sales Effectiveness From Delay Switch → Sales Effectiveness → Orders Booked*

	Loop Description
Loop 14	*Production Capacity → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 15	*Production Capacity → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 16	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 17	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*

	Loop Description
Loop 18	*Production Capacity → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 19	*Production Capacity → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 20	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 21	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*

	Loop Description
Loop 22	*Production Capacity → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 23	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 24	*Production Capacity → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*

	Loop Description
Loop 25	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*

B.4 Market Growth Model Linearly Independent Loops

	Loop Description
Loop 1	*Salesmen → Salesmen Hired*
Loop 2	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Budget → Indicated Salesmen → Salesmen Hired → Salesmen → Salesmen Switch → Orders Booked*
Loop 3	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate*
Loop 4	*Backlog → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Delivery Rate Average Adjustment → Delivery Rate Average → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Recognized By Market Adjustment → Delivery Delay Recognized By Market → Sales Effectiveness From Delay Multiplier → Sales Effectiveness From Delay Switch → Sales Effectiveness → Orders Booked*
Loop 5	*Production Capacity → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 6	*Delivery Rate Average → Delivery Rate Average Adjustment*
Loop 7	*Delivery Delay Recognized By Company → Delivery Delay Recognized By Company Adjustment*
Loop 8	*Delivery Delay Traditional → Delivery Delay Traditional Adjustment*
Loop 9	*Delivery Delay Recognized By Market → Delivery Delay Recognized By Market Adjustment*
Loop 10	*Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2*
Loop 11	*Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3*
Loop 12	*Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 13	*Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Recognized By Market Adjustment → Delivery Delay Recognized By Market → Sales Effectiveness From Delay Multiplier → Sales Effectiveness From Delay Switch → Sales Effectiveness → Orders Booked*

	Loop Description
Loop 14	*Production Capacity → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 15	*Production Capacity → Delivery Delay Minimum → Production Capacity Fraction → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*
Loop 16	*Production Capacity → Delivery Rate → Backlog → Delivery Delay Indicated → Delivery Delay Recognized By Company Adjustment → Delivery Delay Recognized By Company → Delivery Delay Traditional Adjustment → Delivery Delay Traditional → Delivery Delay Operating Goal → Delivery Delay Condition → Capacity Expansion Fraction → Capacity Expansion Fraction Switch → Production Capacity Ordering → Production Capacity Receiving In Transit 1 → Production Capacity Receiving Progress 2 → Production Capacity Receiving In Transit 2 → Production Capacity Receiving Progress 3 → Production Capacity Receiving In Transit 3 → Production Capacity Receiving*

Appendix C

Functions Description

Name	allcycsn
File Name	allcycsn.m
Package	digraph toolbox
Inputs	A (square (Boolean) successor matrix (n,n)): model adjacency matrix
Outputs	cycles (matrix (?,n), each row contains the node numbers in a walk around a cycle; the matrix is padded with 0's on the right)
Description	The algorithm is an exhaustive traversal of the digraph with pruning. An early version is in APL in Evans & Larsen (1981).

Name	allpathn
File Name	allpathn.m
Package	digraph toolbox
Inputs	From (from-node, a number) To (to-node, a number) A (square successor matrix (n,n))
Outputs	allpaths (matrix (?,n), each row contains the node numbers in a walk on a path; the matrix is padded with 0's to the right)
Description	The algorithm is a traversal of the digraph. It uses the reach-ability matrix to prune the traversal.

Name	analysis
File Name	analysis.m
Package	analysis package
Inputs	modelObjectsStructVector modelObjectsValuesMatrix initialTime finalTime timeStep internalStep
Outputs	N/A
Description	The main function in the Analysis package, it calls all the other functions, and executes the steps of the eigenvalue analysis.

Name	computeLinkElasticity
File Name	computeLinkElasticity.m
Package	analysis package

Inputs	numericCompactGainMatrix numericFullGainMatrix modelAdjacencyMatrix modelAdjacencyMatrix2EdgesMatrix rightEigenvectorsMatrix leftEigenvectorsMatrix diagonalEigenvaluesMatrix dominantEigenvaluePosition currentTimeStep
Outputs	numericLinkElasticityVector numericSensitivityByDominantEigenvalueVector tempCheckpoint_2 tempCheckpoint_4 tempCheckpoint_7 tempCheckpoint_8
Description	Computes links elasticity values associated with dominant eigenvalue.

Name	computeSystemJacobians
File Name	computeSystemJacobians.m
Package	analysis package
Inputs	modelObjectsNamesVector modelObjectsEquationsVector constVector constValVector
Outputs	symbolicFullGainMatrix symbolicLinkGain2InputJacobianMatrix modelAdjacencyMatrix modelAdjacencyMatrix2EdgesMatrix
Description	Computes the system Jacobian matrix.

Name	computeIndependentCycleElasticity
File Name	computeIndependentCycleElasticity.m
Package	analysis package
Inputs	independentCyclesEdgesMatrix numericLinkElasticityMatrix
Outputs	independentCyclesElasticityMatrix
Description	Computes independent cycles (loops) elasticity values matrix associated with dominant eigenvalue.

Name	computeInputElasticity
File Name	computeInputElasticity.m
Package	analysis package

Inputs	numericLinkGainVector numericLinkGainVector numericLinkGain2InputJacobianMatrix numericLinkElasticityVector constantsValuesVector numericSensitivityByDominantEigenvalueVector
Outputs	numericInputElasticityVector
Description	Computes inputs elasticity values associated with dominant eigenvalue.

Name	deleteZerosRow
File Name	deleteZerosRow.m
Package	analysis package
Inputs	a (input matrix)
Outputs	res (output matrix)
Description	Removes rows of all zeros in a matrix.

Name	differentiateGraph
File Name	differentiateGraph.m
Package	analysis package
Inputs	input vector output vector input value to find differentiation at
Outputs	out (output differentiation of the graph function at the input point)
Description	The differentiation of the customized interpolation function to suites that one of System Dynamics simulators.

Name	findDominantEigenvalue
File Name	findDominantEigenvalue.m
Package	analysis package
Inputs	rightEigenvectorsMatrix leftEigenvectorsMatrix diagonalEigenvaluesMatrix numericSlopeVector nextNumericSlopeVector levelsValuesVector nextLevelsValuesVector timeStepLength levels2Study currentTimeStep

Outputs	alphasMatrix eigenvaluesMatrix dominantEigenvaluesMatrix dominantEigenvaluesPositionMatrix dominancePercentageMatrix tempCheckpoint_0 tempCheckpoint_1
Description	Finds the dominant eigenvalue.

Name	extractModelObjects
File Name	extractModelObjects.m
Package	analysis package
Inputs	modelObjectsStructVector
Outputs	numStates numAuxiliaries modelObjectsNamesVector modelObjectsEquationsVector
Description	Extracts all objects of the model (names of levels, names of auxiliaries, equations ...) from the vector of structures "modelObjectsStructVector", which comes from the Simulation package.

Name	findIndependentCycles
File Name	findIndependentCycles.m
Package	analysis package
Inputs	modelAdjacencyMatrix modelAdjacencyMatrix2EdgesMatrix modelObjectsNamesVector
Outputs	allCyclesVerticesMatrix independentCyclesVerticesMatrix independentCyclesEdgesMatrix numberIndependentCycles
Description	Finds a set of independent loops, it tries the user selection from the loops of the model and completes them with the shortest set.

Name	jac
File Name	jac.m
Package	analysis package
Inputs	x y
Outputs	out
Description	Computes the Jacobian matrix of two vectors x and y where: out(i,j) = dx(i)/dy(j)

Name	computePathsGain
File Name	computePathsGain.m
Package	analysis package
Inputs	G (gain matrix) paths (matrix (?,n), each row contains the node numbers in a walk around a path; the matrix is padded with 0's on the right)
Outputs	GV (gain vector (1,?))
Description	Calculates paths gains for all given paths in a system

Name	pathgain2
File Name	pathgain2.m
Package	analysis package
Inputs	G (gain matrix) path (matrix (?,n), each row contains the node numbers in a walk around a path; the matrix is padded with 0's on the right) eNode (end node) sNode (start node)
Outputs	GV (gain value)
Description	Calculates path gain for a given path in a system starting at sNode and ends at eNode

Name	printOutputs
File Name	printOutputs.m
Package	analysis package
Inputs	levels2Study modelAdjacencyMatrix internalStep timeStepLength dominantEigenvaluesMatrix dominancePercentageMatrix numericLinkGainMatrix numericLinkElasticityMatrix numericInputElasticityMatrix independentCyclesElasticityMatrix allCyclesVerticesMatrix independentCyclesVerticesMatrix signIndependentCyclesMatrix modelObjectsNamesVector constantsVector outFileName
Outputs	N/A

Description	Prints the outputs of the analysis function as well as saving it to a file called output.out.

Name	reachabi
File Name	reachabi.m
Package	digraph toolbox
Inputs	r (square Boolean reach-ability matrix)
Outputs	M (square matrix)
Description	Reach-ability matrix of the input square matrix m, where: r(i,j) = 1 if node i is reachable from node j; 0 otherwise The matrix m is turned into a Boolean inside.

Appendix D

Variables Description

Name	A12
Type	Matrix
Class	Double
Rows Number	Number of levels
Columns Number	Number of auxiliaries

Name	A21
Type	Matrix
Class	Double
Rows Number	Number of auxiliaries
Columns Number	Number of levels

Name	A22
Type	Matrix
Class	Double
Rows Number	Number of auxiliaries
Columns Number	Number of auxiliaries

Name	BPI
Type	Double
Class	Vector
Rows Number	Number of selected time steps
Columns Number	1

Name	BPI_spans
Type	Double
Class	Vector
Rows Number	Number of found BPI spans
Columns Number	1

Name	I
Type	Double
Class	Scalar

Rows Number	1
Columns Number	1

Name	J_All
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	K
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	allCyclesVerticesMatrix
Type	Double
Class	Matrix
Rows Number	Number of loops
Columns Number	Number of levels + Number of auxiliaries + 1

Name	alphasMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of levels

Name	checkpoint_0
Type	Double
Class	Matrix
Rows Number	Number of selected time steps

Columns Number	Number of levels * 2

Name	checkpoint_1
Type	Double
Class	Matrix
Rows Number	Number of selected time steps
Columns Number	Number of levels * 2

Name	checkpoint_2
Type	Double
Class	Matrix
Rows Number	Number of selected time steps
Columns Number	Number of levels * 2

Name	checkpoint_4
Type	Double
Class	Matrix
Rows Number	Number of selected time steps
Columns Number	(Number of levels + Number of auxiliaries) * 2

Name	checkpoint_7
Type	Double
Class	Matrix
Rows Number	Number of selected time steps
Columns Number	Number of levels * 2

Name	checkpoint_8
Type	Double
Class	Matrix
Rows Number	Number of selected time steps
Columns Number	Number of levels * 2

Name	constantsValuesVector
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of constants

Name	constantsVector
Type	Symbolic
Class	Vector
Rows Number	1
Columns Number	Number of constants

Name	currentTimeStep
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	curvature
Type	Double
Class	Vector
Rows Number	Number of time steps + 2
Columns Number	1

Name	diagonalEigenvaluesMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of levels
Columns Number	Number of levels

Name	dominancePercentageMatrix
Type	Double

Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of levels

Name	dominantEigenvaluesPositionMatrix
Type	Double
Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of levels

Name	dominantEigenvaluesMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of levels

Name	eigenvaluesMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of levels

Name	endLoop
Type	Logical
Class	Scalar
Rows Number	1
Columns Number	1

Name	fid
Type	Double
Class	Scalar
Rows Number	1

Columns Number	1

Name	finalTime
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	independentCyclesEdgesMatrix
Type	Double
Class	Matrix
Rows Number	Number of linearly independent loops
Columns Number	Number of links

Name	independentCyclesElasticityMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of linearly independent loops
Columns Number	Number of time steps

Name	independentCyclesVerticesMatrix
Type	Double
Class	Matrix
Rows Number	Number of linearly independent loops
Columns Number	Number of levels + Number of auxiliaries + 1

Name	initialTime
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	inputs2Study
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of inputs

Name	internalSteps
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of selected time steps

Name	leftEigenvectorsMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of levels
Columns Number	Number of levels

Name	levels2Study
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	max_checkpoint_0
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	max_checkpoint_1
Type	Double
Class	Vector

Rows Number	1
Columns Number	Number of levels * 2

Name	max_checkpoint_2
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	max_checkpoint_4
Type	Double
Class	Vector
Rows Number	1
Columns Number	(Number of levels + Number of auxiliaries) * 2

Name	max_checkpoint_7
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	max_checkpoint_8
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_abs_checkpoint_0
Type	Double
Class	Vector
Rows Number	1

Columns Number	Number of levels * 2

Name	mean_abs_checkpoint_1
Type	Double
Class	Vector
Rows Number	
Columns Number	

Name	mean_abs_checkpoint_2
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_abs_checkpoint_4
Type	Double
Class	Vector
Rows Number	1
Columns Number	(Number of levels + Number of auxiliaries) * 2

Name	mean_abs_checkpoint_7
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_abs_checkpoint_8
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_checkpoint_0
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_checkpoint_1
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_checkpoint_2
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_checkpoint_4
Type	Double
Class	Vector
Rows Number	1
Columns Number	(Number of levels + Number of auxiliaries) * 2

Name	mean_checkpoint_7
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	mean_checkpoint_8
Type	Double

Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	min_checkpoint_0
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	min_checkpoint_1
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	min_checkpoint_2
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	min_checkpoint_4
Type	Double
Class	Vector
Rows Number	1
Columns Number	(Number of levels + Number of auxiliaries) * 2

Name	min_checkpoint_7
Type	Double
Class	Vector
Rows Number	1

Columns Number	Number of levels * 2

Name	min_checkpoint_8
Type	Double
Class	Vector
Rows Number	1
Columns Number	Number of levels * 2

Name	modelAdjacencyMatrix
Type	Double
Class	Matrix
Rows Number	Number of levels + Number of auxiliaries
Columns Number	Number of levels + Number of auxiliaries

Name	modelAdjacencyMatrix2EdgesMatrix
Type	Double
Class	Matrix
Rows Number	Number of levels + Number of auxiliaries
Columns Number	Number of levels + Number of auxiliaries

Name	modelObjectsEquationsVector
Type	Symbolic
Class	Vector
Rows Number	1
Columns Number	Number of levels + Number of auxiliaries

Name	modelObjectsNamesVector
Type	Symbolic
Class	Vector
Rows Number	1
Columns Number	Number of levels + Number of auxiliaries

Name	modelObjectsStructVector
Type	Structure
Class	Vector
Rows Number	1
Columns Number	Number of levels + Number of auxiliaries

Name	modelObjectsValuesMatrix
Type	Double
Class	Matrix
Rows Number	Number of time steps + 2
Columns Number	Number of levels + Number of auxiliaries

Name	netflowsValuesMatrix
Type	Double
Class	Matrix
Rows Number	Number of time steps + 2
Columns Number	Number of levels

Name	numAuxiliaries
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	numLevels
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	numLinks
Type	Double

Class	Scalar
Rows Number	1
Columns Number	1

Name	numTimeSteps
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	numberIndependentCycles
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Name	numericCompactGainMatrix
Type	Double
Class	Matrix
Rows Number	Number of levels
Columns Number	Number of levels

Name	numericFullGainMatrix
Type	Double
Class	Matrix
Rows Number	Number of levels + Number of auxiliaries
Columns Number	Number of levels + Number of auxiliaries

Name	numericInputElasticityMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of inputs

Columns Number	Number of time steps

Name	numericLinkElasticityMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of links
Columns Number	Number of time steps

Name	numericLinkGain2InputJacobianMatrix
Type	Double
Class	Matrix
Rows Number	Number of links
Columns Number	Number of inputs

Name	numericLinkGainMatrix
Type	Double
Class	Matrix
Rows Number	Number of links
Columns Number	Number of time steps

Name	numericLinkSensitivityByDominantEigenvalueMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of links
Columns Number	Number of time steps

Name	outFileName
Type	Char
Class	Vector
Rows Number	1
Columns Number	Output file name length

Name	rightEigenvectorsMatrix
Type	Double (Complex)
Class	Matrix
Rows Number	Number of levels
Columns Number	Number of levels

Name	signIndependentCyclesMatrix
Type	Double
Class	Matrix
Rows Number	Number of time steps
Columns Number	Number of linearly independent loops

Name	suggestedInternalStep
Type	Double
Class	Vector
Rows Number	Number of selected time steps
Columns Number	1

Name	symbolicFullGainMatrix
Type	Symbolic
Class	Matrix
Rows Number	Number of levels + Number of auxiliaries
Columns Number	Number of levels + Number of auxiliaries

Name	symbolicLinkGain2InputJacobianMatrix
Type	Symbolic
Class	Matrix
Rows Number	Number of links
Columns Number	Number of inputs

Name	tempCheckpoint_0
Type	Double

Class	Matrix
Rows Number	2
Columns Number	Number of levels * 2

Name	tempCheckpoint_1
Type	Double
Class	Matrix
Rows Number	2
Columns Number	Number of levels * 2

Name	tempCheckpoint_2
Type	Double
Class	Matrix
Rows Number	2
Columns Number	Number of levels * 2

Name	tempCheckpoint_4
Type	Double
Class	Matrix
Rows Number	2
Columns Number	Number of levels + Number of auxiliaries

Name	tempCheckpoint_7
Type	Double
Class	Matrix
Rows Number	2
Columns Number	Number of levels * 2

Name	*tempCheckpoint_8*
Type	Double
Class	Matrix
Rows Number	2

Columns Number	Number of levels * 2

Name	*tempNumericCompactGainMatrix*
Type	Symbolic
Class	Matrix
Rows Number	Number of levels
Columns Number	Number of levels

Name	*tempSymbolicFullGainMatrix*
Type	Symbolic
Class	Matrix
Rows Number	Number of levels + Number of auxiliaries
Columns Number	Number of levels + Number of auxiliaries

Name	*tempSymbolicLinkGain2InputJacobianMatrix*
Type	Double
Class	Matrix
Rows Number	Number of links
Columns Number	Number of inputs

Name	*timeStepLength*
Type	Double
Class	Scalar
Rows Number	1
Columns Number	1

Appendix E

Internal Functions

The following are the full listings of the Analysis package functions on Matlab 6.5.

E.1 analysis.m

It is the main function in the Analysis package; it calls all the other functions, and executes the steps of the eigenvalue analysis.

```
function analysis( modelObjectsStructVector , constantsVector ,
constantsValuesVector
, modelObjectsValuesMatrix , netflowsValuesMatrix , initialTime ,
finalTime , timeSt
epLength , outFileName )

% ---------------------------------------------------------------
---------
% Filename: analysis.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: modelObjectsStructVector
% modelObjectsValuesMatrix
% initialTime
% finalTime
% timeStepLength
% internalSteps
% Outputs: N/A
% Description: The main function in the Analysis package, it
calls all the
% other functions, and executes the steps of the eigenvalue
% analysis
% ---------------------------------------------------------------
----------

disp( [ sprintf('\n') 'Starting Analysis' ]);

% Extract Model Objects
[ numLevels , numAuxiliaries , modelObjectsNamesVector ,
modelObjectsEquationsVector
] = ...
extractModelObjects( modelObjectsStructVector );

% Empty and Initialize checkpoint (0)
fid = fopen( [ 'checkpoint_0.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"findDominantEigenvalue.m" at the
end of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'it computes the error (E) and percentage error
(PE)' sprintf('\n') ]
);
fwrite( fid , [ 'between the absolute value of:' sprintf('\n') ]
);
fwrite( fid , [ 'next time step State Vector X(t+1),'
```

```
sprintf('\n') ] );
fwrite( fid , [ 'the one comes from simulation' sprintf('\n') ]
);
fwrite( fid , [ 'and the computed one from the (alpha / lambda) *
exp(lambda * dt) eq
uations ...' sprintf('\n\n') ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
fwrite( fid , [ 'E (X' num2str( I ) ');PE (X' num2str( I ) ');' ]
);
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );
% Empty and Initialize checkpoint (1)
fid = fopen( [ 'checkpoint_1.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"findDominantEigenvalue.m" at the
end of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'it computes the error (E) and percentage error
(PE)' sprintf('\n') ]
);
fwrite( fid , [ 'between the absolute value of:' sprintf('\n') ]
);
fwrite( fid , [ 'next time step Slope Vector X dot(t+1),'
sprintf('\n') ] );
fwrite( fid , [ 'the one comes from simulation' sprintf('\n') ]
);
fwrite( fid , [ 'and the computed one from the alpha * exp(lambda
* dt) equations ...
' sprintf('\n\n') ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
fwrite( fid , [ 'E (X' num2str( I ) ''');PE (X' num2str( I )
''');' ] );
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );
% Empty and Initialize checkpoint (2)
fid = fopen( [ 'checkpoint_2.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"computeLinkElasticity.m" at the e
nd of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'for every time step:' sprintf('\n') ] );
fwrite( fid , [ 'The error between sum of row( i ) and column( i
) of' sprintf('\n')]
);
fwrite( fid , [ 'the compact Elasticity values matrix, also the
percentage error (PE)
.' sprintf('\n')] );
fwrite( fid , [ '(they should be the same, for any Level the
Elasticity value enterin
g' sprintf('\n') ] );
fwrite( fid , [ 'should be the same value leaving (Forrester, N.,
1983))' sprintf('\n
\n') ] );
```

```
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
fwrite( fid , [ 'E (r&c' num2str( I ) ');PE (r&c' num2str( I )
');' ] );
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );

% Empty and Initialize checkpoint (3)
fid = fopen( [ 'checkpoint_3.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"computeLinkElasticity.m" at the e
nd of the file,' sprintf('\n')] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'for every time step:' sprintf('\n') ] );
fwrite( fid , [ 'The sum of all elements of the compact
Elasticity values matrix' spr
intf('\n') ] );
fwrite( fid , [ '(should eqaul 1)' sprintf('\n\n') ] );
fwrite( fid , [ 'Time;SUM(E)' sprintf('\n\n') ] );
fclose( fid );

% Empty and Initialize checkpoint (4)
fid = fopen( [ 'checkpoint_4.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"computeLinkElasticity.m" at the e
nd of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'for every time step:' sprintf('\n') ] );
fwrite( fid , [ 'The error between sum of row( i ) and column( i
) of' sprintf('\n')
] );
fwrite( fid , [ 'the full Elasticity values matrix, also the
percentage error (PE).'
sprintf('\n') ] );
fwrite( fid , [ '(they should be the same, for any variable
(Level or Auxiliary) the'
sprintf('\n') ] );
fwrite( fid , [ 'Elasticity value entering should be the same
value leaving' sprintf(
'\n\n') ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels + numAuxiliaries,
fwrite( fid , [ 'E (r&c' num2str( I ) ');PE (r&c' num2str( I )
');' ] );
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );

% Empty and Initialize checkpoint (5)
fid = fopen( [ 'checkpoint_5.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by
"computeIndependentCycleElasticity
.m" at the end of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'numericLinkElasticityMatrixComputed = Cr *
independentCyclesElastici
tyMatrix' sprintf('\n') ] );
```

```
fwrite( fid , [ 'error (E) = numericLinkElasticityMatrix - 
numericLinkElasticityMatri
xComputed' sprintf('\n') ] );
fwrite( fid , [ 'and also the percentage error (PE) ...' 
sprintf('\n\n') ] );
fclose( fid );

% Empty and Initialize checkpoint (7)
fid = fopen( [ 'checkpoint_7.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by 
"computeLinkElasticity.m" at the e
nd of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'for every time step:' sprintf('\n') ] );
fwrite( fid , [ 'The error between sum of col( i ) and col( i ) 
of' sprintf('\n') ] )
;
fwrite( fid , [ 'the Elasticity values matrix and the full 
Elasticity values matrix r
espectively,' sprintf('\n') ] );
fwrite( fid , [ 'also the percentage error (PE).' sprintf('\n') 
] );
fwrite( fid , [ '(they should be the same, for all Levels, the' 
sprintf('\n') ] );
fwrite( fid , [ 'Elasticity value entering any Level in both 
matrices should be the s
ame value' sprintf('\n\n') ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
fwrite( fid , [ 'E (r&c' num2str( I ) ');PE (r&c' num2str( I ) 
');' ] );
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );

% Empty and Initialize checkpoint (8)
fid = fopen( [ 'checkpoint_8.csv' ] , 'w' );
fwrite( fid , [ 'This checkpoint file generated by 
"computeLinkElasticity.m" at the e
nd of the file,' sprintf('\n') ] );
fwrite( fid , [ 'it contains the following:' sprintf('\n') ] );
fwrite( fid , [ 'for every time step:' sprintf('\n') ] );
fwrite( fid , [ 'The error between sum of row( i ) and row( i ) 
of' sprintf('\n') ] )
;
fwrite( fid , [ 'the Elasticity values matrix and the full 
Elasticity values matrix r
espectively,' sprintf('\n') ] );
fwrite( fid , [ 'also the percentage error (PE).' sprintf('\n') 
] );
fwrite( fid , [ '(they should be the same, for all Levels, the' 
sprintf('\n') ] );
fwrite( fid , [ 'Elasticity value leaving any Level in both 
matrices should be the sa
me value' sprintf('\n\n') ] );
fwrite( fid , [ 'Time;' ] );
for I = 1 : numLevels,
fwrite( fid , [ 'E (r&c' num2str( I ) ');PE (r&c' num2str( I ) 
');' ] );
```

```
end
fwrite( fid , [ sprintf('\n\n') ] );
fclose( fid );

% Time Steps
numTimeSteps = ( ( finalTime - initialTime ) / timeStepLength )
+ 1;

% Which level to study its behavior?
endLoop = true;
while ( endLoop )
for I = 1 : numLevels,
disp( [ int2str( I ) ' - ' char( modelObjectsNamesVector( I ) )
] );
end
levels2Study = input( [ 'Enter the number of the level, you are
intersted' sprint
f('\n') 'in studying (ex.: 2):' sprintf('\t') ] );
if length( levels2Study ) ~= 1 | levels2Study > numLevels |
levels2Study < 1,
disp( 'Wrong Input(s), try again ...' );
else
endLoop = false;
end
end

% Which constants are inputs?
endLoop = true;
while ( endLoop )
for I = 1 : length( constantsVector ),
disp( [ int2str( I ) ' - ' char( constantsVector( I ) ) ] );
end
inputs2Study = input( [ 'Enter the number of constants, you
would like to' sprint
f('\n') 'consider as inputs (ex.: 30 or [ 1,2,3 ] or [ 1:50 ] ):'
sprintf('\t') ] );
if isempty( inputs2Study ),
inputs2Study = [ 1 : length( constantsVector ) ];
endLoop = false;
elseif inputs2Study > length( constantsVector ) | max(
inputs2Study ) > length( c
onstantsVector ) | min(inputs2Study) < 1,
disp( 'Wrong Input(s), try again ...' );
else
endLoop = false;
end
end

% suggesting time steps to study according to Behavior Pattern
Index
curvature = zeros( size( netflowsValuesMatrix( : , levels2Study
) ) );
curvature( 2:end ) = diff( netflowsValuesMatrix( : ,
levels2Study ) ) / timeStepLengt
h;
BPI = sign( curvature( : ) ./ netflowsValuesMatrix( : ,
levels2Study ) );
```

```
if isnan( BPI( end ) ),
endLoop = true;
J = [ length( BPI ) ];
I = length( BPI ) - 1;
while( endLoop ),
if ~isnan( BPI( I ) ),
endLoop = false;
else
J = [ J , I ];
I = I - 1;
end
end
BPI( J ) = BPI( I );
end
endLoop = true;
J_All = find( isnan( BPI ) );
for K = J_All.',
if isnan( BPI( K ) ),
J = K;
I = K + 1;
while( endLoop ),
if ~isnan( BPI( I ) ),
endLoop = false;
else
J = [ J , I ];
I = I + 1;
end
end
BPI( J ) = BPI( I );
end
end
BPI_spans = diff( BPI );
BPI_spans=[ 1 ; find( abs( BPI_spans( : ) ) == 2 ) ; numTimeSteps ];
suggestedInternalStep = BPI_spans + [ round( diff( BPI_spans ) / 2 ) ; 0 ];
suggestedInternalStep( end ) = [];

% Plot the level selected to study
plot( modelObjectsValuesMatrix( 1 : numTimeSteps , levels2Study ) , 'LineWidth' , 2 ) ;
set( gca , 'XTick' , BPI_spans );
set( gca , 'XTickLabel' , { num2str( round( ( BPI_spans - 1 ) * timeStepLength * 10 ) / 10 ) } );
set( gca , 'XGrid' , 'on' );
axis tight;
xlabel( 'time' );
title( char( modelObjectsNamesVector( levels2Study ) ) );

% Which time steps to study the selected level behavior at?
endLoop = true;
while ( endLoop )
disp( [ 'Time Steps range is from 1 to ' int2str( numTimeSteps ) ] );
```

```
disp( [ 'Corresponding to Time Instants range from' num2str( (
initialTime - 1 )
* timeStepLength ) ' to ' num2str( ( numTimeSteps - 1 ) *
timeStepLength ) ] );
disp( [ 'it is suggested to do analysis at the following time
steps: ' sprintf('\
n') int2str( suggestedInternalStep.' ) ] );
disp( [ 'Corresponding to following time instants: '
sprintf('\n') num2str( [ ( s
uggestedInternalStep - 1 ) * timeStepLength ].' ) ] );
disp( [ 'Corresponding to Time Instants range from 1 to '
num2str( ( I - 1 ) * ti
meStepLength ) ] );
internalSteps = input( [ 'Enter the time steps, you are
intersted' sprintf('\n')
'in studying (ex.: 30 or [ 1,2,3 ] or [ 1:50 ] ):' sprintf('\t') ] );
if isempty( internalSteps ),
internalSteps = [ 1 : numTimeSteps ];
endLoop = false;
elseif max( internalSteps ) > numTimeSteps | min(internalSteps)
< 1,
disp( 'Wrong Input(s), try again ...' );
else
endLoop = false;
end
end

% Variables Initializations
checkpoint_0 = [];
checkpoint_1 = [];
checkpoint_2 = [];
checkpoint_4 = [];
% % % % % % checkpoint_6 = [];
checkpoint_7 = [];
checkpoint_8 = [];

dominantEigenvaluesMatrix = zeros( numTimeSteps , numLevels );
dominantEigenvaluesPositionMatrix = zeros( numTimeSteps ,
numLevels );
dominancePercentageMatrix = zeros( numTimeSteps , numLevels );
% Compute Jacobians of the model
[ symbolicFullGainMatrix , symbolicLinkGain2InputJacobianMatrix
, modelAdjacencyMatri
x , modelAdjacencyMatrix2EdgesMatrix ] = ...
computeSystemJacobians( modelObjectsNamesVector ,
modelObjectsEquationsVector , c
onstantsVector , constantsValuesVector , inputs2Study );

% Variables Initializations
numLinks = max( max( modelAdjacencyMatrix2EdgesMatrix ) );
numericLinkElasticityMatrix = zeros( numLinks , numTimeSteps );
numericLinkSensitivityByDominantEigenvalueMatrix = zeros(
numLinks , numTimeSteps );
numericLinkGainMatrix = zeros( numLinks , numTimeSteps );
numericInputElasticityMatrix = zeros( length( inputs2Study ) ,
numTimeSteps );

% Finding Set Independent Loop
```

```
[ allCyclesVerticesMatrix , independentCyclesVerticesMatrix ,
independentCyclesEdgesM
atrix , numberIndependentCycles ] = ...
findIndependentCycles( modelAdjacencyMatrix ,
modelAdjacencyMatrix2EdgesMatrix ,
modelObjectsNamesVector );
% Variables Initializations
signIndependentCyclesMatrix = zeros( numTimeSteps ,
numberIndependentCycles );
% 'for loop' of the selected analysis time steps
for currentTimeStep = internalSteps ,
disp( [ 'Step: ' , num2str( currentTimeStep ) ' of: ' , num2str(
numTimeSteps ) ]
);
% Compute Numeric Full Gain Matrix and Numeric Link Gain to
Input Jacobian Matrix
tempSymbolicLinkGain2InputJacobianMatrix = ...
subs( symbolicLinkGain2InputJacobianMatrix , sym('TIME') ,
currentTimeStep );
numericLinkGain2InputJacobianMatrix = ...
double( subs( tempSymbolicLinkGain2InputJacobianMatrix ,
modelObjectsNamesVec
tor , modelObjectsValuesMatrix( currentTimeStep , : ) ) );
tempSymbolicFullGainMatrix = ...
subs( symbolicFullGainMatrix , sym('TIME') , ( currentTimeStep *
timeStepLeng
th ) + initialTime );
numericFullGainMatrix = ...
double( subs( tempSymbolicFullGainMatrix ,
modelObjectsNamesVector , modelObj
ectsValuesMatrix( currentTimeStep , : ) ) );
% Polarity of Independent Cycles
signIndependentCyclesMatrix( currentTimeStep , : ) = sign(
computePathsGain( nume
ricFullGainMatrix , independentCyclesVerticesMatrix ) );
[ x , y ] = find( modelAdjacencyMatrix2EdgesMatrix ~= 0 );
for I = 1 : length( x ),
numericLinkGainMatrix( modelAdjacencyMatrix2EdgesMatrix( x( I )
, y( I ) ) ,
currentTimeStep ) = ...
numericFullGainMatrix( x( I ) , y( I ) );
end
% The Compact Model Gain Matrix
% [ A11 A12 ]
% [ ]
% [ A21 A22 ]
%
% m * n *
% m m
%
% m * n *
% n n
```

```
%
% m = length( levelsVector )
% n = length( auxiliariesVector )
% Note: A11 will always be a null matrix
A12 = numericFullGainMatrix( 1 : numLevels , numLevels+1 : end );
A21 = numericFullGainMatrix( numLevels+1 : end , 1 : numLevels );
A22 = numericFullGainMatrix( numLevels+1 : end , numLevels+1 : end );
numericCompactGainMatrix = A12 * inv( eye( size( A22 ) ) - A22 ) * A21;

% Computing the eigenvalues and eigenvectors of the Compact Gain Matrix
tempNumericCompactGainMatrix = sym( numericCompactGainMatrix , 'd' );
[ rightEigenvectorsMatrix , diagonalEigenvaluesMatrix ] = eig( tempNumericCompactGainMatrix );
rightEigenvectorsMatrix = double( rightEigenvectorsMatrix );
diagonalEigenvaluesMatrix = double( diagonalEigenvaluesMatrix );
leftEigenvectorsMatrix = inv( rightEigenvectorsMatrix ).';

% Finding the dominant eigenvalue
[ dominantEigenvaluesMatrix( currentTimeStep , : ) , dominantEigenvaluesPositionMatrix( currentTimeStep , : ) , dominancePercentageMatrix( currentTimeStep , : ) , tempCheckpoint_0 , tempCheckpoint_1 ] = ...
findDominantEigenvalue( rightEigenvectorsMatrix , leftEigenvectorsMatrix , diagonalEigenvaluesMatrix , netflowsValuesMatrix( currentTimeStep , : ).' , netflowsValuesMatrix( currentTimeStep + 1 , : ).' , modelObjectsValuesMatrix( currentTimeStep , 1 : numLevels ).' , modelObjectsValuesMatrix( currentTimeStep + 1 , 1 : numLevels ).' , timeStepLength , levels2Study , currentTimeStep );
checkpoint_0 = [ checkpoint_0 ; tempCheckpoint_0(:).' ];
checkpoint_1 = [ checkpoint_1 ; tempCheckpoint_1(:).' ];

% Computing the symbolic links elasticity values associated with the
% dominant eigenvalue
[ numericLinkElasticityMatrix( : , currentTimeStep ) , numericLinkSensitivityByDominantEigenvalueMatrix( : , currentTimeStep ) , tempCheckpoint_2 , tempCheckpoint_4 , tempCheckpoint_7 , tempCheckpoint_8 ] = ...
computeLinkElasticity( numericCompactGainMatrix , numericFullGainMatrix , modelAdjacencyMatrix , modelAdjacencyMatrix2EdgesMatrix , rightEigenvectorsMatrix , leftEigenvectorsMatrix , diagonalEigenvaluesMatrix , dominantEigenvaluesPositionMatrix( currentTimeStep , 1 ) , currentTimeStep );
checkpoint_2 = [ checkpoint_2 ; tempCheckpoint_2(:).' ];
checkpoint_4 = [ checkpoint_4 ; tempCheckpoint_4(:).' ];
checkpoint_7 = [ checkpoint_7 ; tempCheckpoint_7(:).' ];
```

```
checkpoint_8 = [ checkpoint_8 ; tempCheckpoint_8(:).' ];

% Computing the symbolic inputs elasticity values associated with the
% dominant eigenvalue
[ numericInputElasticityMatrix( : , currentTimeStep ) ] = ...
computeInputElasticity( numericLinkGainMatrix( : , currentTimeStep ) , numericLinkGain2InputJacobianMatrix , numericLinkElasticityMatrix( : , currentTimeStep ) , constantsValuesVector , inputs2Study , numericLinkSensitivityByDominantEigenvalueMatrix( : , currentTimeStep ) );

end

% Computing independent cycles elasticity values associated with dominant
% eigenvalue
independentCyclesElasticityMatrix = ...
computeIndependentCycleElasticity( independentCyclesEdgesMatrix , numericLinkElasticityMatrix );

% end checkpoint (0)
mean_checkpoint_0 = mean( checkpoint_0 );
mean_abs_checkpoint_0 = mean( abs( checkpoint_0 ) );
max_checkpoint_0 = max( checkpoint_0 );
min_checkpoint_0 = min( checkpoint_0 );
fid = fopen( [ 'checkpoint_0.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_checkpoint_0( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_abs_checkpoint_0( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( max_checkpoint_0( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( min_checkpoint_0( I ) ) ';' ] );
end
fclose( fid );

% end checkpoint (1)
mean_checkpoint_1 = mean( checkpoint_1 );
mean_abs_checkpoint_1 = mean( abs( checkpoint_1 ) );
max_checkpoint_1 = max( checkpoint_1 );
min_checkpoint_1 = min( checkpoint_1 );
fid = fopen( [ 'checkpoint_1.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * numLevels,
```

```
fwrite( fid , [ num2str( mean_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_abs_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( max_checkpoint_1( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( min_checkpoint_1( I ) ) ';' ] );
end
fclose( fid );

% end checkpoint (2)
mean_checkpoint_2 = mean( checkpoint_2 );
mean_abs_checkpoint_2 = mean( abs( checkpoint_2 ) );
max_checkpoint_2 = max( checkpoint_2 );
min_checkpoint_2 = min( checkpoint_2 );
fid = fopen( [ 'checkpoint_2.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_checkpoint_2( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_abs_checkpoint_2( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( max_checkpoint_2( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( min_checkpoint_2( I ) ) ';' ] );
end
fclose( fid );

% end checkpoint (4)
mean_checkpoint_4 = mean( checkpoint_4 );
mean_abs_checkpoint_4 = mean( abs( checkpoint_4 ) );
max_checkpoint_4 = max( checkpoint_4 );
min_checkpoint_4 = min( checkpoint_4 );
fid = fopen( [ 'checkpoint_4.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * ( numLevels + numAuxiliaries ),
fwrite( fid , [ num2str( mean_checkpoint_4( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * ( numLevels + numAuxiliaries ),
fwrite( fid , [ num2str( mean_abs_checkpoint_4( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
```

```
for I = 1 : 2 * ( numLevels + numAuxiliaries ),
fwrite( fid , [ num2str( max_checkpoint_4( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * ( numLevels + numAuxiliaries ),
fwrite( fid , [ num2str( min_checkpoint_4( I ) ) ';' ] );
end
fclose( fid );

% end checkpoint (7)
mean_checkpoint_7 = mean( checkpoint_7 );
mean_abs_checkpoint_7 = mean( abs( checkpoint_7 ) );
max_checkpoint_7 = max( checkpoint_7 );
min_checkpoint_7 = min( checkpoint_7 );
fid = fopen( [ 'checkpoint_7.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_checkpoint_7( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_abs_checkpoint_7( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( max_checkpoint_7( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( min_checkpoint_7( I ) ) ';' ] );
end
fclose( fid );

% end checkpoint (8)
mean_checkpoint_8 = mean( checkpoint_8 );
mean_abs_checkpoint_8 = mean( abs( checkpoint_8 ) );
max_checkpoint_8 = max( checkpoint_8 );
min_checkpoint_8 = min( checkpoint_8 );
fid = fopen( [ 'checkpoint_8.csv' ] , 'a' );
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_checkpoint_8( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( mean_abs_checkpoint_8( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( max_checkpoint_8( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * numLevels,
fwrite( fid , [ num2str( min_checkpoint_8( I ) ) ';' ] );
end
```

```
fclose( fid );

% Printing to output file
printOutputs( levels2Study , inputs2Study , modelAdjacencyMatrix
, internalSteps , ti
meStepLength , dominantEigenvaluesMatrix , dominancePercentageMatrix
, numericLinkGai
nMatrix , numericLinkElasticityMatrix , numericInputElasticityMatrix
, independentCyc
lesElasticityMatrix , allCyclesVerticesMatrix ,
independentCyclesVerticesMatrix , sig
nIndependentCyclesMatrix , modelObjectsNamesVector , constantsVector
, outFileName );

disp( [ sprintf( '\n' ) 'Finishing Analysis' ] );
```

E.2 computeIndependentCycleElasticity.m

It computes independent cycles (loops) elasticity values matrix associated with dominant eigenvalue.

```
function independentCyclesElasticityMatrix =
computeIndependentCycleElasticity( indep
endentCyclesEdgesMatrix , numericLinkElasticityMatrix )

% ---------------------------------------------------------------
---------
% Filename: computeIndependentCycleElasticity.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: independentCyclesEdgesMatrix
% numericLinkElasticityMatrix
% Outputs: independentCyclesElasticityMatrix
% Description: Computes independent cycles (loops) elasticity
values
% matrix associated with dominant eigenvalue
% ---------------------------------------------------------------
----------

disp( [ 'Computing Independent Cycle Elasticity' ]);

% [ k1 ] [ l1 ]
% [ k2 ] [ l2 ]
% [ . ] = Cr * [ . ]
% [ . ] [ lm ]
% [ kn ]
%
% ki: links , lj:loops
%
Cr = independentCyclesEdgesMatrix.';

```

```
% A least squares solution is computed
independentCyclesElasticityMatrix = Cr \
numericLinkElasticityMatrix;
warning off MATLAB:divideByZero;
% checkpoint (5)
numericLinkElasticityMatrixComputed = Cr *
independentCyclesElasticityMatrix;
E = abs( numericLinkElasticityMatrix -
numericLinkElasticityMatrixComputed );
PE = abs( 100 * E ./ numericLinkElasticityMatrix );
fid = fopen( [ 'checkpoint_5.csv' ] , 'a' );
fwrite( fid , [ 'Time;' ] );
for I = 1 : size( E , 1 ),
fwrite( fid , [ 'E (lnk' num2str( I ) ''');PE (lnk' num2str( I )
''');' ] );
end
fwrite( fid , [ sprintf('\n\n') ] );
for J = 1 : size( E , 2 ),
fwrite( fid , [ num2str( J ) ';' ] );
for I = 1 : size( E , 1 ),
fwrite( fid , [ num2str( E( I , J ) ) ';' ] );
fwrite( fid , [ num2str( PE( I , J ) ) ';' ] );
end
fwrite( fid , sprintf('\n') );
end
mean_checkpoint_5 = [ mean(E.') ; mean(PE.') ];
mean_checkpoint_5 = mean_checkpoint_5(:).';
mean_abs_checkpoint_5 = [ mean(abs(E.')) ; mean(abs(PE.')) ];
mean_abs_checkpoint_5 = mean_abs_checkpoint_5(:).';
max_checkpoint_5 = [ max(E.') ; max(PE.') ];
max_checkpoint_5 = max_checkpoint_5(:).';
min_checkpoint_5 = [ min(E.') ; min(PE.') ];
min_checkpoint_5 = min_checkpoint_5(:).';
fwrite( fid , [ sprintf('\n') 'Mean;' ] );
for I = 1 : 2 * size( E , 1 ),
fwrite( fid , [ num2str( mean_checkpoint_5( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Mean Abs.;' ] );
for I = 1 : 2 * size( E , 1 ),
fwrite( fid , [ num2str( mean_abs_checkpoint_5( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Max;' ] );
for I = 1 : 2 * size( E , 1 ),
fwrite( fid , [ num2str( max_checkpoint_5( I ) ) ';' ] );
end
fwrite( fid , [ sprintf('\n') 'Min;' ] );
for I = 1 : 2 * size( E , 1 ),
fwrite( fid , [ num2str( min_checkpoint_5( I ) ) ';' ] );
```

```
end
fclose( fid );
```

E.3 computeInputElasticity.m

It computes inputs elasticity values associated with dominant eigenvalue.

```
function [ numericInputElasticityVector ] =
computeInputElasticity( numericLinkGainVe
ctor , numericLinkGain2InputJacobianMatrix ,
numericLinkElasticityVector , constantsV
aluesVector , inputs2Study ,
numericLinkSensitivityByDominantEigenvalueVector )

% ---------------------------------------------------------------
---------
% Filename: computeInputElasticity.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: numericLinkGainVector
% numericLinkGainVector
% numericLinkGain2InputJacobianMatrix
% numericLinkElasticityVector
% constantsValuesVector
% inputs2Study
% numericLinkSensitivityByDominantEigenvalueVector
% Outputs: numericInputElasticityVector
% Description: Computes inputs elasticity values associated with
% dominant eigenvalue
% ---------------------------------------------------------------
----------

disp( [ 'Computing Input Elasticity' ]);

% warning off MATLAB:divideByZero;

% Variables Initializations
numInputs = length( inputs2Study );
numericInputElasticityVector = zeros( numInputs , 1 );

% Inputs elasticity values vector calculation
for I = 1 : numInputs,
numericInputElasticityVector( I ) = ...
constantsValuesVector( I ) * sum(
numericLinkGain2InputJacobianMatrix( : , I
) .* numericLinkSensitivityByDominantEigenvalueVector );
end
```

E.4 computeLinkElasticity.m

It computes links elasticity values associated with dominant eigenvalue.

```
function [ numericLinkElasticityVector ,
numericLinkSensitivityByDominantEigenvalueVe
ctor , tempCheckpoint_2 , tempCheckpoint_4 , tempCheckpoint_7 ,
tempCheckpoint_8 ] =
computeLinkElasticity( numericCompactGainMatrix ,
numericFullGainMatrix , modelAdjace
ncyMatrix , modelAdjacencyMatrix2EdgesMatrix ,
rightEigenvectorsMatrix , leftEigenvec
torsMatrix , diagonalEigenvaluesMatrix , dominantEigenvaluePosition ,
currentTimeStep
)

% -----------------------------------------------------------------
---------
% Filename: computeLinkElasticity.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: numericCompactGainMatrix
% numericFullGainMatrix
% modelAdjacencyMatrix
% modelAdjacencyMatrix2EdgesMatrix
% rightEigenvectorsMatrix
% leftEigenvectorsMatrix
% diagonalEigenvaluesMatrix
% dominantEigenvaluePosition
% currentTimeStep
% Outputs: numericLinkElasticityVector
% numericLinkSensitivityByDominantEigenvalueVector
% tempCheckpoint_2
% tempCheckpoint_4
% tempCheckpoint_7
% tempCheckpoint_8
% Description: Computes links elasticity values associated with
% dominant eigenvalue
% ----------------------------------------------------------------
----------

disp( [ 'Computing Link Elasticity' ]);

% Variables Initializations
numLevels = size( numericCompactGainMatrix , 1 );
numLinks = max( max( modelAdjacencyMatrix2EdgesMatrix ) );
numericLinkElasticityVector = zeros( numLinks , 1 );
numericLinkSensitivityByDominantEigenvalueVector = zeros(
numLinks , 1 );
numericFullElasticityMatrix = zeros( size( numericFullGainMatrix
) );
numericFullSensitivityByDominantEigenvalueMatrix = zeros( size(
numericFullGainMatrix
) );

% Sensitivity and elasticity associated with dominant eigenvalue
values
numericSensitivityMatrix = ...
leftEigenvectorsMatrix( : , dominantEigenvaluePosition ) *
rightEigenvectorsMatri
x( : , dominantEigenvaluePosition ).';
```

```

numericLinkSensitivityByDominantEigenvalueMatrix = ...
numericSensitivityMatrix / diagonalEigenvaluesMatrix(
dominantEigenvaluePosition
, dominantEigenvaluePosition );

numericElasticityMatrix = ...
numericLinkSensitivityByDominantEigenvalueMatrix .*
numericCompactGainMatrix;

% The Full Elasticity Values Matrix
[ x , y ] = find( numericCompactGainMatrix ~= 0 );
for I = 1 : length(x),
% Find all paths that starts at y(I) and ends at x(I)
pathsMatrix = allpathn( y( I ) , x( I ) , modelAdjacencyMatrix );

% Deleting paths that pass through a level
w = [];
for K = 1 : size( pathsMatrix , 1 ) ,
path = nonzeros( pathsMatrix( K , : ) );
if any( path( 2 : end - 1 ) <= numLevels ),
w = [ w K ];
end
end
pathsMatrix( w , : ) = [];

% Computing the kth path gain
pathsGainsVector = computePathsGain( numericFullGainMatrix ,
pathsMatrix );

% the Elasticity value of the kth path ( from I to J )
pathsElasticityValuesVector = pathsGainsVector *
numericLinkSensitivityByDominant
EigenvalueMatrix( x( I ) , y( I ) );

% The gain and Elasticity value of the kth path from y(I) to x(I)
for K = 1 : size( pathsMatrix , 1 ) ,
% the kth path from y(I) to x(I)
path = nonzeros( pathsMatrix( K , : ) );

% for each element in the path
for J = 1 : length( path ) - 1 ,
numericLinkElasticityVector( modelAdjacencyMatrix2EdgesMatrix(
path( J+1
) , path( J ) ) ) = ...
numericLinkElasticityVector( modelAdjacencyMatrix2EdgesMatrix(
path(
J+1 ) , path( J ) ) ) + pathsElasticityValuesVector(K);

numericFullElasticityMatrix( path( J+1 ) , path( J ) ) = ...
numericFullElasticityMatrix( path( J+1 ) , path( J ) ) +
pathsElastic
ityValuesVector( K );


tempPathGain = computePathsGain2( numericFullGainMatrix , path ,
```

```
path( J+
1 ) , path( J ) );
tempPathSensitivityByDominantEigenvalueMatrix = tempPathGain *
numericLin
kSensitivityByDominantEigenvalueMatrix( x( I ) , y( I ) );
numericFullSensitivityByDominantEigenvalueMatrix( path( J+1 ) ,
path( J )
) = ...
numericFullSensitivityByDominantEigenvalueMatrix( path( J+1 ) ,
path(
J ) ) + tempPathSensitivityByDominantEigenvalueMatrix;
numericLinkSensitivityByDominantEigenvalueVector(
modelAdjacencyMatrix2Ed
gesMatrix( path( J+1 ) , path( J ) ) ) = ...
numericLinkSensitivityByDominantEigenvalueVector(
modelAdjacencyMatri
x2EdgesMatrix( path( J+1 ) , path( J ) ) ) +
tempPathSensitivityByDominantEigenvalueM
atrix;
end
end
end
warning off MATLAB:divideByZero;
% checkpoint (2, 3, 4)
E1 = abs( sum( numericElasticityMatrix , 1 ) - [ sum(
numericElasticityMatrix , 2 ) ]
.' );
PE1 = abs( 100 * E1 ./ sum( numericElasticityMatrix , 1 ) );
E2 = abs( sum( numericFullElasticityMatrix , 1 ) - [ sum(
numericFullElasticityMatrix
, 2 ) ].' );
PE2 = abs( 100 * E2 ./ sum( numericFullElasticityMatrix , 1 ) );
colNumericFullElasticityMatrix=numericFullElasticityMatrix( : ,
1 : numLevels );
rowNumericFullElasticityMatrix=numericFullElasticityMatrix( 1 :
numLevels , : );
E3i = abs( sum( colNumericFullElasticityMatrix , 1 ) - sum(
numericElasticityMatrix ,
1 ) );
E3o = abs( sum( rowNumericFullElasticityMatrix , 2 ) - sum(
numericElasticityMatrix ,
2 ) );
PE3i = abs( 100 * E3i ./ sum( colNumericFullElasticityMatrix , 1
) );
PE3o = abs( 100 * E3o ./ sum( rowNumericFullElasticityMatrix , 2
) );
tempCheckpoint_2 = [ E1(:).' ; PE1(:).' ];
tempCheckpoint_4 = [ E2(:).' ; PE2(:).' ];
tempCheckpoint_7 = [ E3i(:).' ; PE3i(:).' ];
tempCheckpoint_8 = [ E3o(:).' ; PE3o(:).' ];
```

```

fid = fopen( [ 'checkpoint_2.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : size( numericElasticityMatrix , 1 ),
fwrite( fid , [ num2str( E1( I ) ) ';' ] );
fwrite( fid , [ num2str( PE1( I ) ) ';' ] );
end
fwrite( fid , sprintf( '\n' ) );
fclose( fid );

fid = fopen( [ 'checkpoint_3.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' num2str( abs(
sum( sum( numericElastic
ityMatrix ) ) ) ) sprintf( '\n' ) ] );
fclose( fid );

fid = fopen( [ 'checkpoint_4.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : size( numericFullElasticityMatrix , 1 ),
fwrite( fid , [ num2str( E2( I ) ) ';' ] );
fwrite( fid , [ num2str( PE2( I ) ) ';' ] );
end
fwrite( fid , sprintf( '\n' ) );
fclose( fid );

fid = fopen( [ 'checkpoint_7.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : size( numericElasticityMatrix , 1 ),
fwrite( fid , [ num2str( E3i( I ) ) ';' ] );
fwrite( fid , [ num2str( PE3i( I ) ) ';' ] );
end
fwrite( fid , sprintf( '\n' ) );
fclose( fid );

fid = fopen( [ 'checkpoint_8.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : size( numericElasticityMatrix , 1 ),
fwrite( fid , [ num2str( E3o( I ) ) ';' ] );
fwrite( fid , [ num2str( PE3o( I ) ) ';' ] );
end
fwrite( fid , sprintf( '\n' ) );
fclose( fid );
```

E.5 computePathsGain.m

It calculates paths gains for all given paths in a system.

```
function GV = computePathsGain( G , paths )

% -------------------------------------------------------------
---------
% Filename: computePathsGain.m
% Author: Ahmed AbdelTawab AbdelGawad
```

```
% Package: Analysis Package
% Inputs: G (gain matrix)
% paths (matrix (?,n), each row contains the node numbers in
% a walk around a path; the matrix is padded with 0's on the
% right)
% Outputs: GV (gain vector (1,?))
% Description: Calculates paths gains for all given paths in a system
% ----------------------------------------------------------------------

GV = ones( 1 , size( paths , 1 ) );

for i = 1 : size( paths , 1 ),
cn = paths( i , 1 : max( find( paths( i , : ) ) ) );
for j = 1 : length( cn ) - 1,
GV( i ) = GV( i ) * G( cn( j + 1 ) , cn( j ) );
end
end
```

E.6 computePathsGain2.m

It calculates path gain for a given path in a system starting at start node and ends at end node.

```
function GV = computePathsGain2( G , path , eNode , sNode )

% ---------------------------------------------------------------------
% Filename: computePathsGain2.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: G (gain matrix)
% path (matrix (?,n), each row contains the node numbers in
% a walk around a path; the matrix is padded with 0's on the
% right)
% eNode (end node)
% sNode (start node)
% Outputs: GV (gain value)
% Description: path gain for a given path in a system excluding
% one link that starting at sNode and ends at eNode
% ----------------------------------------------------------------------

GV = 1;

cn = path( 1 : max( find( path ) ) );
for j = 1 : length( cn ) - 1,
if ~( ( cn( j + 1 ) == eNode ) & ( cn( j ) == sNode ) )
GV = GV * G( cn( j+1 ),cn( j ) );
end
end
```

E.7 computeSystemJacobians.m

It computes the system Jacobian matrix, links gain to input Jacobian matrix, model adjacency matrix and model adjacency matrix to edges matrix.

```
function [ symbolicFullGainMatrix , symbolicLinkGain2InputJacobianMatrix , modelAdjacencyMatrix , modelAdjacencyMatrix2EdgesMatrix ] = computeSystemJacobians( modelObjectsNamesVector , modelObjectsEquationsVector , constantsVector , constantsValuesVector , inputs2Study )

% -----------------------------------------------------------------------
% Filename: computeSystemJacobians.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: modelObjectsNamesVector
% modelObjectsEquationsVector
% constantsVector
% constantsValuesVector
% inputs2Study
% Outputs: symbolicFullGainMatrix
% symbolicLinkGain2InputJacobianMatrix
% modelAdjacencyMatrix
% modelAdjacencyMatrix2EdgesMatrix
% Description: Computes the system Jacobian matrix, links gain to
% input Jacobian matrix, model adjacency matrix and model
% adjacency matrix to edges matrix
% ------------------------------------------------------------------------

disp( [ 'Compute System Jacobians (Symbolic)' ]);

% symbolicFullGainMatrix
symbolicFullGainMatrix = ...
jac( modelObjectsEquationsVector.' , modelObjectsNamesVector );

% modelAdjacencyMatrix
modelAdjacencyMatrix = ...
zeros( size( symbolicFullGainMatrix ) );
modelAdjacencyMatrix( find( symbolicFullGainMatrix ~= 0 ) ) = 1;

% modelAdjacencyMatrix2EdgesMatrix
modelAdjacencyMatrix2EdgesMatrix = ...
modelAdjacencyMatrix;
modelAdjacencyMatrix2EdgesMatrix( find( modelAdjacencyMatrix ~= 0 ) ) = ...
[ 1 : nnz( modelAdjacencyMatrix ) ];

% symbolicLinkGain2InputJacobianMatrix
```

```
[ x , y ] = find( modelAdjacencyMatrix2EdgesMatrix ~= 0 );
for I = 1 : length( x ),
symbolicLinkGainVector( modelAdjacencyMatrix2EdgesMatrix( x( I )
, y( I ) ) ) = ...
symbolicFullGainMatrix( x( I ) , y( I ) );
end
symbolicLinkGain2InputJacobianMatrix = ...
jac( symbolicLinkGainVector , constantsVector( inputs2Study ) );

symbolicLinkGain2InputJacobianMatrix = ...
subs( symbolicLinkGain2InputJacobianMatrix , constantsVector ,
constantsValuesVec
tor );

symbolicFullGainMatrix = ...
subs( symbolicFullGainMatrix , constantsVector ,
constantsValuesVector );
```

E.8 deleteZerosRow.m

It removes rows of all zeros in a matrix.

```
function res = deleteZerosRow( a )

% ---------------------------------------------------------------
---------
% Filename: deleteZerosRow.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: a
% Outputs: res
% Description: Removes rows of all zeros in a matrix
% ---------------------------------------------------------------
----------

res = a;

if isempty( a )
return;
end

if size( a , 2 ) > 1,
res = a( find( sum( a.' ) ) , : );
else
res = a( find( a ~= 0 ) );
end
```

E.9 differentiateGraph.m

The differentiation of the customized interpolation function to suites that one of System Dynamics simulators.

```
function out = differentiateGraph( inp , varargin )

% -------------------------------------------------------------------------
% Filename: differentiateGraph.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: Input Vector
% Output Vector
% Input Value to find differentiation at
% Outputs: out (Output)
% Description: The differentiation of the customized interpolation
% function to suites that one of System Dynamics simulators
% --------------------------------------------------------------------------


% differentiateGraph(inp,x,y)
n = (nargin - 1) / 2;
x = varargin( 1 : n );
y = varargin( n + 1 : end );

x = cell2num(x);
y = cell2num(y);

inp = subs( inp );

if isnumeric( inp ),
[ x , IX ] = sort( x );
y = y( IX );

% Find indices of subintervals, x( k ) <= inp < x( k + 1 ),
% or inp < x( 1 ) or inp >= x( end ).

k = sum( x < inp ); % 0 ---> n
if k == 0,
% Extrapolate
if inp == x( 1 ),
out = ( ( y( 2 ) - y( 1 ) ) / ( x( 2 ) - x( 1 ) ) ) / 2;
else
out = 0;
end
elseif k == n,
% Extrapolate
if inp == x( end ),
out = ( ( y( end ) - y( end -1 ) ) / ( x( end ) - x( end - 1 ) ) ) / 2;
else
out = 0;
end
else
% Interpolate
if inp == x( k ),
out = mean( [ ( y( k + 1 ) - y( k ) ) / ( x( k + 1 ) - x( k ) ) ,
```

```
( y( k
) - y( k - 1 ) ) / ( x( k ) - x( k - 1 ) ) ] );
else
out = ( y( k + 1 ) - y( k ) ) / ( x( k + 1 ) - x( k ) ) ;
end
end
else
out = sym( [ 'differentiateGraph(' char( inp ) ',' rowv( x ) ','
rowv( y ) ')' ]
);
end

%----------------------
function v = rowv(x)
v = sym(x);
v = char(v(:).');
v([1:8 end-2:end]) = [];

% ----------------------
function v = cell2num(x)
v = zeros(size(x));
for n = 1 : length( x )
v( n ) = x{ n };
end
```

E.10 extractModelObjects.m

It extracts all objects of the model (names of levels, names of auxiliaries, equations ...) from the vector of structures *modelObjectsStructVector*, which comes from the Simulation package.

```
function [ numLevels , numAuxiliaries , modelObjectsNamesVector ,
modelObjectsEquatio
nsVector ] = extractModelObjects( modelObjectsStructVector )

% -------------------------------------------------------------
---------
% Filename: extractModelObjects.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: modelObjectsStructVector
% Outputs: numLevels
% numAuxiliaries
% modelObjectsNamesVector
% modelObjectsEquationsVector
% Description: Extracts all objects of the model (names of
levels, names
% of auxiliaries, equations ...) form the vector of
% structures "modelObjectsStructVector", which comes from the
% Simulation package
% -------------------------------------------------------------
```

```
----------

disp( [ sprintf('\n') 'Extracting Model Objects' ]);

% Compute number of levels and auxiliaries
numLevels = sum( [ modelObjectsStructVector.state ] );
numAuxiliaries = sum( ~[ modelObjectsStructVector.state ] );

% Extract objects of the model
modelObjectsNamesVector = [ modelObjectsStructVector.name ];
modelObjectsEquationsVector = [ modelObjectsStructVector.equation
];
```

E.11 findDominantEigenvalue.m

It finds the dominant eigenvalue.

```
function [ dominantEigenvaluesVector ,
dominantEigenvaluesPositionVector , dominanceP
ercentageVector , tempCheckpoint_0 , tempCheckpoint_1 ] =
findDominantEigenvalue( rig
htEigenvectorsMatrix , leftEigenvectorsMatrix ,
diagonalEigenvaluesMatrix , netflowsV
aluesVector , nextNetflowsValuesVector , levelsValuesVector ,
nextLevelsValuesVector
, timeStepLength , levels2Study , currentTimeStep )

% ----------------------------------------------------------------
---------
% Filename: findDominantEigenvalue.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: rightEigenvectorsMatrix
% leftEigenvectorsMatrix
% diagonalEigenvaluesMatrix
% netflowsValuesVector
% nextNetflowsValuesVector
% levelsValuesVector
% nextLevelsValuesVector
% timeStepLength
% levels2Study
% currentTimeStep
% Outputs: dominantEigenvaluesVector
% dominantEigenvaluesPositionVector
% dominancePercentageVector
% tempCheckpoint_0
% tempCheckpoint_1
% Description: Finds the dominant eigenvalue
% ---------------------------------------------------------------
----------

disp( [ 'Finding Dominant Eigenvalue' ]);

% Time step used in the analysis process
```

```
analysisTimeStepLength = timeStepLength;

% Initializations
numLevels = size( diagonalEigenvaluesMatrix , 1 );
eigenvaluesVector = diag( diagonalEigenvaluesMatrix ).';
dominantEigenvaluesVector = eigenvaluesVector ;
dominancePercentageVector = [];

% At (t - tao) = 0 --> alphasVector = initial alphasVector and levelsValuesVector = initial value of slope
alphasVector = leftEigenvectorsMatrix.' * netflowsValuesVector;

deltaStateTerms = ( zeros( size( rightEigenvectorsMatrix ) ) );
deltaSlopeTerms = ( zeros( size( rightEigenvectorsMatrix ) ) );

for K = 1 : numLevels ,
if eigenvaluesVector( K ) == 0,
deltaStateTerms( : , K ) = rightEigenvectorsMatrix( : , K ) .* ( alphasVector ( K ) * analysisTimeStepLength );
else
deltaStateTerms( : , K ) = rightEigenvectorsMatrix( : , K ) .* ( alphasVector ( K ) * ( exp( eigenvaluesVector( K ) * analysisTimeStepLength ) - 1 ) / eigenvaluesVector( K ) );
end
end

deltaSlopeTerms = rightEigenvectorsMatrix * ( alphasVector .* ( exp( eigenvaluesVector( : ) * analysisTimeStepLength ) - 1 ) );

deltaState = sum( deltaStateTerms , 2 );
deltaSlope = sum( deltaSlopeTerms , 2 );

for K = 1 : numLevels ,
flags = zeros( numLevels , 1 );
flags( K ) = 1;
if ~isreal( eigenvaluesVector( K ) ),
conjK = find( eigenvaluesVector == conj( eigenvaluesVector( K ) ) );
flags( conjK ) = 1;
end
deltaStateTerm = sum( deltaStateTerms( levels2Study , : ) .* flags.' );
contribution = deltaStateTerm / deltaState( levels2Study );
dominancePercentageVector = [ dominancePercentageVector , 100 * real( contribution ) ];
end

[ dominancePercentageVector , dominantEigenvaluesPositionVector ] = sort( dominancePercentageVector );
dominancePercentageVector = fliplr( dominancePercentageVector );
```

```
dominantEigenvaluesPositionVector = fliplr(
dominantEigenvaluesPositionVector );
dominantEigenvaluesVector = dominantEigenvaluesVector(
dominantEigenvaluesPositionVec
tor );
alphasVector = alphasVector.';
% checkpoint (0)
numericNextTimeStateVector = ...
deltaState + levelsValuesVector;
E = abs( nextLevelsValuesVector - numericNextTimeStateVector );
PE = abs( 100 * E ./ nextLevelsValuesVector );
tempCheckpoint_0 = [ E(:).' ; PE(:).' ];
fid = fopen( [ 'checkpoint_0.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : length( numericNextTimeStateVector ),
fwrite( fid , [ num2str( E( I ) ) ';' ] );
fwrite( fid , [ num2str( PE( I ) ) ';' ] );
end
fwrite( fid , sprintf('\n') );
fclose( fid );
% checkpoint (1)
numericNextTimeSlopeHatVector = ...
deltaSlope + netflowsValuesVector ;
E = abs( nextNetflowsValuesVector - numericNextTimeSlopeHatVector
);
PE = abs( 100 * E ./ nextNetflowsValuesVector );
tempCheckpoint_1 = [ E(:).' ; PE(:).' ];
fid = fopen( [ 'checkpoint_1.csv' ] , 'a' );
fwrite( fid , [ num2str( currentTimeStep ) ';' ] );
for I = 1 : length( numericNextTimeSlopeHatVector ),
fwrite( fid , [ num2str( E( I ) ) ';' ] );
fwrite( fid , [ num2str( PE( I ) ) ';' ] );
end
fwrite( fid , sprintf('\n') );
fclose( fid );
```

E.12 findIndependentCycles.m

It finds a set of independent loops; it tries the user selection from the loops of the model and completes them with the shortest set.

```
function [ allCyclesVerticesMatrix , independentCyclesVerticesMatrix
, independentCyc
lesEdgesMatrix , numberIndependentCycles ] = findIndependentCycles(
modelAdjacencyMat
rix , modelAdjacencyMatrix2EdgesMatrix , modelObjectsNamesVector )
% -----------------------------------------------------------------
------
% Filename: findIndependentCycles.m
```

```
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: modelAdjacencyMatrix
% modelAdjacencyMatrix2EdgesMatrix
% modelObjectsNamesVector
% Outputs: allCyclesVerticesMatrix
% independentCyclesVerticesMatrix
% independentCyclesEdgesMatrix
% numberIndependentCycles
% Description: Finds a set of independent loops, it tries the user
% selection from the loops of the model and completes them
% with the shortest set
% ---------------------------------------------------------------
-------

disp( [ 'Finding Set Independent Loop' ]);

% all cycles
allCyclesVerticesMatrix = allcycsn( modelAdjacencyMatrix );
numberCycles = size( allCyclesVerticesMatrix , 1 );

% the Cycles' matrix (all cycles expressed in binary form by links)
allCyclesEdgesMatrix = zeros( numberCycles , max( max (
modelAdjacencyMatrix2EdgesMat
rix ) ) );
for I = 1 : numberCycles,
oneCycle = nonzeros( allCyclesVerticesMatrix( I , : ) ).';
for J = 1 : size( oneCycle , 2 ) - 1,
K = modelAdjacencyMatrix2EdgesMatrix( oneCycle( J + 1 ) , oneCycle(
J ) );
allCyclesEdgesMatrix( I , K ) = 1;
end;
end;

% Which loops to start searching for an independent set with?
endLoop = true;
while ( endLoop ),
% Printing All Loops
disp( [ sprintf('\n') 'All Loops:' sprintf('\n') ] );
for I = 1:size( allCyclesVerticesMatrix , 1 ) ,
tempPrint = [ ];
oneIndependentCycle = nonzeros( allCyclesVerticesMatrix( I , : )
).';
disp( [ 'Loop' sprintf('\t') num2str( I ) ':' sprintf('\t') ] );
for J = 1:size( oneIndependentCycle , 2 )-1 ,
tempPrint = [ tempPrint , char( modelObjectsNamesVector(
oneIndependentCy
cle( J ) ) ) ];
if J ~= size( oneIndependentCycle , 2 )-1 ,
tempPrint = [ tempPrint , ' --> ' ];
end
end
disp( tempPrint );
end
loops2Study = input( [ 'Enter the number(s) of the Loop(s) you are
intersted' spr
```

```
intf('\n') 'in studying in a vector form (ex.: [1,2,6]):' sprintf('\t')
] );
if max( loops2Study ) > size( allCyclesVerticesMatrix , 1 ) | min(
loops2Study )
< 1 | size( loops2Study , 1 ) ~= 1,
disp('Wrong Input(s), try again ...');
else
endLoop = false;
end
end
numberIndependentCycles = rank( allCyclesEdgesMatrix );
independentCyclesEdgesMatrix = allCyclesEdgesMatrix;
independentCyclesVerticesMatrix = allCyclesVerticesMatrix;
temp1 = independentCyclesEdgesMatrix( loops2Study , : );
temp2 = independentCyclesVerticesMatrix( loops2Study , : );
independentCyclesEdgesMatrix( loops2Study , : ) = [];
independentCyclesVerticesMatrix( loops2Study , : ) = [];
independentCyclesEdgesMatrix = [temp1;independentCyclesEdgesMatrix];
independentCyclesVerticesMatrix =
[temp2;independentCyclesVerticesMatrix];
independentCyclesEdgesMatrix = flipud( independentCyclesEdgesMatrix
);
independentCyclesVerticesMatrix = flipud(
independentCyclesVerticesMatrix );
for I = 1:numberCycles,
tempCycles = independentCyclesEdgesMatrix;
tempCyclesn = independentCyclesVerticesMatrix;
independentCyclesEdgesMatrix( I , : ) = 0;
independentCyclesVerticesMatrix( I , : ) = 0;
if ~( rank( independentCyclesEdgesMatrix ) ==
numberIndependentCycles ),
independentCyclesEdgesMatrix = tempCycles;
independentCyclesVerticesMatrix = tempCyclesn;
end;
end;
independentCyclesVerticesMatrix = flipud(
deleteZerosRow(independentCyclesVerticesMat
rix ) );
independentCyclesEdgesMatrix = flipud(
deleteZerosRow(independentCyclesEdgesMatrix )
);
```

E.13 jac.m

It computes the Jacobian matrix of two input vectors.

```
function out = jac(x,y)

% ------------------------------------------------------------------------
% Filename: jac.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: x
% y
% Outputs: out
% Description: Computes the Jacobian matrix of two vectors x and y, where:
% out(i,j) = dx(i)/dy(j)
% ------------------------------------------------------------------------

out = sym( [ ] );
for I = 1:length( x ),
for J = 1:length( y ),
out( I , J ) = differentiate( x( I ) , y( J ) );
end
end

% --------------------------------------
function R = differentiate( S , a )

% ------------------------------------------------------------------------
% Filename: jac.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: S
% a
% Outputs: R
% Description: Computes the differentiation dS/da
% ------------------------------------------------------------------------

S = sym( S );
a = sym( a );
str = char(S);
if strncmp( str , 'ifthenelse' , 10 ),
ix1 = strfind( str , '(' );
ix2 = strfind( str , ',' );
ix3 = strfind( str , ')' );
var0 = str( 1 : ix1( 1 ) - 1 );
var1 = str( ix1( 1 ) + 1 : ix2( 1 ) - 1 );
var2 = str( ix2( 1 ) + 1 : ix2( 2 ) - 1 );
var3 = sym( str( ix2( 2 ) + 1 : ix2( 3 ) - 1 ) );
var4 = sym( str( ix2( 3 ) + 1 : ix3( 1 ) - 1 ) );

var3 = maple('map','diff',var3,a);
var4 = maple('map','diff',var4,a);

if var3 == sym(0) & var4 == sym(0),
```

```
R = sym(0);
else
R = sym( [ var0 '(' var1 ',' var2 ',' char( var3 ) ',' char( var4 ) ')' ] );
end

elseif strncmp( str , 'graph' , 5 ),
ix1 = strfind( str , '(' );
ix2 = strfind( str , ',' );
var1 = sym( str( ix1( 1 ) + 1 : ix2( 1 ) - 1 ) );
var2 = maple('map','diff',var1,a);
R = sym( strrep( str , 'graph' , 'differentiateGraph' ) ) * var2;

else
R = maple('map','diff',S,a);
end
```

E.14 printOutputs.m

It prints the outputs of the analysis function as well as saving it to a file.

```
function printOutputs( levels2Study , inputs2Study , modelAdjacencyMatrix , internalSteps , timeStepLength , dominantEigenvaluesMatrix , dominancePercentageMatrix , numericLinkGainMatrix , numericLinkElasticityMatrix , numericInputElasticityMatrix , independentCyclesElasticityMatrix , allCyclesVerticesMatrix , independentCyclesVerticesMatrix , signIndependentCyclesMatrix , modelObjectsNamesVector , constantsVector , outFileName )

% -------------------------------------------------------------------------
% Filename: printOutputs.m
% Author: Ahmed AbdelTawab AbdelGawad
% Package: Analysis Package
% Inputs: levels2Study
% inputs2Study
% modelAdjacencyMatrix
% internalSteps
% timeStepLength
% dominantEigenvaluesMatrix
% dominancePercentageMatrix
% numericLinkGainMatrix
% numericLinkElasticityMatrix
% numericInputElasticityMatrix
% independentCyclesElasticityMatrix
% allCyclesVerticesMatrix
% independentCyclesVerticesMatrix
% signIndependentCyclesMatrix
% modelObjectsNamesVector
% constantsVector
% outFileName
```

```
% Outputs: N/A
% Description: Prints the outputs of the analysis function as well as
% saving it to a file called output.out
% --------------------------------------------------------------------------

disp( [ sprintf( '\n' ) 'Priniting Outputs' ] );

% Empty the output file
fid = fopen( outFileName , 'w' );

% Printing all eigenvalues and their dominance percentage
fwrite( fid , [ 'The eigenvalues and their dominance percentage contribution to the level variable ''' char( modelObjectsNamesVector( levels2Study ) ) ''':' ] );
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength ) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '__________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1 : length( dominantEigenvaluesMatrix( I , : ) ) ,
fwrite( fid , [ num2str( dominantEigenvaluesMatrix( I , J ) ) ', with percentage contribution: ' int2str( dominancePercentageMatrix( I , J ) ) '%.' ] );
fwrite( fid , [ sprintf('\n') ] );
end
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

[ to , from ] = find( modelAdjacencyMatrix );
% Printing the Links and the Links Gains
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All links and their gains:' ] );
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength ) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '__________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1 : size( numericLinkGainMatrix , 1 ) ,
fwrite( fid , [ char( modelObjectsNamesVector( from( J ) ) ) ' --> ' char( mo
```

```
delObjectsNamesVector( to( J ) ) ) ': ' num2str( (
numericLinkGainMatrix( J , I ) )
) ] );
fwrite( fid , [ sprintf('\n') ] );
end
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%
dominantEigenvaluesMatrix = dominantEigenvaluesMatrix( : , 1 );
dominancePercentageMatrix = dominancePercentageMatrix( : , 1 );
% Printing the Links and the Links' elasticity values
dummy_linkElasticity_Sorted = zeros( size(
numericLinkElasticityMatrix ) );
dummy_linkElasticity_Sorted_I = zeros( size(
numericLinkElasticityMatrix ) );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All links and their elasticity values to the
dominant eigenvalue:' ]
);
tempPrint = { };
tempPrint_I = { };
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '__________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
% % % % % % % % % % % % % % % % % % % %
if isreal( dominantEigenvaluesMatrix( I ) ),
for J = 1 : size( numericLinkElasticityMatrix , 1 ) ,
fwrite( fid , [ char( modelObjectsNamesVector( from( J ) ) ) ' -->
' char
( modelObjectsNamesVector( to( J ) ) ) ': ' num2str( real(
numericLinkElasticityMatr
ix( J , I ) ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
tempPrint{ J , I } = [ char( modelObjectsNamesVector( from( J ) ) )
' -->
' char( modelObjectsNamesVector( to( J ) ) ) ': ' num2str( real(
numericLinkElastici
tyMatrix( J , I ) ) ) ];
dummy_linkElasticity_Sorted( J , I ) = real(
```

```
numericLinkElasticityMatrix(
J , I ) );
end
else
for J = 1 : size( numericLinkElasticityMatrix , 1 ) ,
fwrite( fid , [ char( modelObjectsNamesVector( from( J ) ) ) ' -->
' char
( modelObjectsNamesVector( to( J ) ) ) ': ' num2str( (
numericLinkElasticityMatrix(
J , I ) ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
tempPrint{ J , I } = [ char( modelObjectsNamesVector( from( J ) ) )
' -->
' char( modelObjectsNamesVector( to( J ) ) ) ': ' num2str( real(
numericLinkElastici
tyMatrix( J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix(
I ) ) ) ) ];
tempPrint_I{ J , I } = [ char( modelObjectsNamesVector( from( J ) )
) ' -
-> ' char( modelObjectsNamesVector( to( J ) ) ) ': ' num2str( imag(
numericLinkElasti
cityMatrix( J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix
( I ) ) ) ) ];
dummy_linkElasticity_Sorted( J , I ) = real(
numericLinkElasticityMatrix(
J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix( I ) ) );
dummy_linkElasticity_Sorted_I( J , I ) = imag(
numericLinkElasticityMatri
x( J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix( I ) ) )
;
end
end
% % % % % % % % % % % % % % % % % % % % 
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

% Printing the Links and the Links' elasticity values (Sorted)
[ dummy_linkElasticity_Sorted , IX ] = sort(
dummy_linkElasticity_Sorted , 1 );
[ dummy_linkElasticity_Sorted_I , IX_I ] = sort(
dummy_linkElasticity_Sorted_I , 1 );
IX = flipud( IX );
IX_I = flipud( IX_I );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All Links and their elasticity values to the
dominant eigenvalue (So
rted):' ] );
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
```

```
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ 'Effect on the Envelope:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
end
for J = 1:size( numericLinkElasticityMatrix , 1 ) ,
fwrite( fid , tempPrint{ IX( J , I ) , I } );
fwrite( fid , [ sprintf('\n') ] );
end
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Effect on the Frequency:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1:size( numericLinkElasticityMatrix , 1 ) ,
fwrite( fid , tempPrint_I{ IX_I( J , I ) , I } );
fwrite( fid , [ sprintf('\n') ] );
end
end
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_________________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

% Printing the Inputs and their elasticity values
dummy_InputElasticity_Sorted = zeros( size(
numericInputElasticityMatrix ) );
dummy_InputElasticity_Sorted_I = zeros( size(
numericInputElasticityMatrix ) );

fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All inputs and their elasticity values to the
dominant eigenvalue:'
] );
tempPrint = { };
tempPrint_I = { };
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
```

```
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
% % % % % % % % % % % % % % % % % % % % %
if isreal( dominantEigenvaluesMatrix( I ) ),
for J = 1 : length( inputs2Study ),
fwrite( fid , [ char( constantsVector( inputs2Study( J ) ) ) ': '
num2str
( real( numericInputElasticityMatrix( J , I ) ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
tempPrint{ J , I } = [ char( constantsVector( inputs2Study( J ) ) )
': '
num2str( real( numericInputElasticityMatrix( J , I ) ) ) ];
dummy_InputElasticity_Sorted( J , I ) = real(
numericInputElasticityMatri
x( J , I ) );
end
else
for J = 1 : length( inputs2Study ),
fwrite( fid , [ char( constantsVector( inputs2Study( J ) ) ) ': '
num2str
( ( numericInputElasticityMatrix( J , I ) ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
tempPrint{ J , I } = [ char( constantsVector( inputs2Study( J ) ) )
': '
num2str( real( numericInputElasticityMatrix( J , I ) *
dominantEigenvaluesMatrix( I )
/ abs( dominantEigenvaluesMatrix( I ) ) ) ) ];
tempPrint_I{ J , I } = [ char( constantsVector( inputs2Study( J ) )
) ':
' num2str( imag( numericInputElasticityMatrix( J , I ) *
dominantEigenvaluesMatrix( I
) / abs( dominantEigenvaluesMatrix( I ) ) ) ) ];
dummy_InputElasticity_Sorted( J , I ) = real(
numericInputElasticityMatri
x( J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix( I ) ) )
;
dummy_InputElasticity_Sorted_I( J , I ) = imag(
numericInputElasticityMat
rix( J , I ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvaluesMatrix( I ) )
);
end
end
% % % % % % % % % % % % % % % % % % % % % %
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%
```

```
% Printing the Inputs and their elasticity values (Sorted)
[ dummy_InputElasticity_Sorted , IX ] = sort(
dummy_InputElasticity_Sorted , 1 );
[ dummy_InputElasticity_Sorted_I , IX_I ] = sort(
dummy_InputElasticity_Sorted_I , 1
);
IX = flipud( IX );
IX_I = flipud( IX_I );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All inputs and their elasticity values to the
dominant eigenvalue (S
orted):' ] );
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ 'Effect on the Envelope:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
end
for J = 1:size( numericInputElasticityMatrix , 1 ) ,
fwrite( fid , tempPrint{ IX( J , I ) , I } ); %IX(:,1)
fwrite( fid , [ sprintf('\n') ] );
end
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Effect on the Frequency:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1:size( numericInputElasticityMatrix , 1 ) ,
fwrite( fid , tempPrint_I{ IX_I( J , I ) , I } );
fwrite( fid , [ sprintf('\n') ] );
end
end
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

% Printing All Loops...
```

```
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'All loops:' ] );
for I = 1:size( allCyclesVerticesMatrix , 1 ) ,
tempPrint = [ ];
oneCycle = nonzeros( allCyclesVerticesMatrix( I , : ) ).';
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Loop ' num2str( I ) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1:size( oneCycle , 2 )-1 ,
tempPrint = [ tempPrint , char( modelObjectsNamesVector( oneCycle(
J ) ) ) ];
if J ~= size( oneCycle , 2 )-1 ,
tempPrint = [ tempPrint , ' --> ' ];
end
end
fwrite( fid , tempPrint );
fwrite( fid , [ sprintf('\n') ] );
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

% Printing Independent Loops...
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Linearly independent loops:' ] );
fwrite( fid , [ sprintf('\n') ] );
for I = 1:size( independentCyclesVerticesMatrix , 1 ) ,
tempPrint = [ ];
oneCycle = nonzeros( independentCyclesVerticesMatrix( I , : ) ).';
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Loop ' num2str( I ) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for J = 1:size( oneCycle , 2 )-1 ,
tempPrint = [ tempPrint , char( modelObjectsNamesVector( oneCycle(
J ) ) ) ];
if J ~= size( oneCycle , 2 )-1 ,
tempPrint = [ tempPrint , ' --> ' ];
end
end
fwrite( fid , tempPrint );
fwrite( fid , [ sprintf('\n') ] );
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'_______________________________________________________' ] );
```

```
fwrite( fid , [ sprintf('\n') ] );
%

% Printing the Loops' elasticity values

fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Independent loops'' elasticity values:' ] );
independentCyclesElasticityMatrix =
independentCyclesElasticityMatrix.';
dummy_IndependentCyclesElasticity_Sorted = zeros( size(
independentCyclesElasticityMa
trix ) );
dummy_IndependentCyclesElasticity_Sorted_I = zeros( size(
independentCyclesElasticity
Matrix ) );

for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '___________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
% % % % % % % % % % % % % % % % % % % %
if isreal( dominantEigenvaluesMatrix( I ) ),
for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
fwrite( fid , [ 'Loop ' num2str( K ) ' (Polarity: ' num2str(
signIndepend
entCyclesMatrix( I , K ) ) '): ' num2str( real(
independentCyclesElasticityMatrix( I
, K ) ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
dummy_IndependentCyclesElasticity_Sorted( I , K ) = real(
independentCycl
esElasticityMatrix( I , K ) );
end
else
for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
fwrite( fid , [ 'Loop ' num2str( K ) ' (Polarity: ' num2str(
signIndepend
entCyclesMatrix( I , K ) ) '): ' num2str(
independentCyclesElasticityMatrix( I , K )
) ] );
fwrite( fid , [ sprintf('\n') ] );
dummy_IndependentCyclesElasticity_Sorted( I , K ) = real(
independentCycl
esElasticityMatrix( I , K ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenvalu
esMatrix( I ) ) );
dummy_IndependentCyclesElasticity_Sorted_I( I , K ) = imag(
independentCy
```

```
clesElasticityMatrix( I , K ) * dominantEigenvaluesMatrix( I ) / abs(
dominantEigenva
luesMatrix( I ) ) );
end
end
% % % % % % % % % % % % % % % % % % % %
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );
%

% Printing the Loops' elasticity values ( Sorted )
[ dummy_IndependentCyclesElasticity_Sorted , IX ] = sort(
dummy_IndependentCyclesElas
ticity_Sorted , 2 );
[ dummy_IndependentCyclesElasticity_Sorted_I , IX_I ] = sort(
dummy_IndependentCycles
Elasticity_Sorted_I , 2 );
dummy_IndependentCyclesElasticity_Sorted = fliplr(
dummy_IndependentCyclesElasticity_
Sorted );
dummy_IndependentCyclesElasticity_Sorted_I = fliplr(
dummy_IndependentCyclesElasticit
y_Sorted_I );
IX = fliplr( IX );
IX_I = fliplr( IX_I );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Independent loops'' elasticity values (Sorted):' ]
);
for I = internalSteps ,
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Time instant ' num2str( ( I - 1 ) * timeStepLength
) ':' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ '__________________' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'The dominant eigenvalue is: ' num2str(
dominantEigenvaluesMatrix
( I ) ) ', with percentage contribution: ' int2str(
dominancePercentageMatrix( I ) )
'%.' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ 'Effect on the Envelope:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
end
for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
fwrite( fid , [ 'Loop ' num2str( IX( I , K ) ) ' (Polarity: '
num2str( signIn
dependentCyclesMatrix( I , IX( I , K ) ) ) '): ' num2str(
dummy_IndependentCyclesElas
ticity_Sorted( I , K ) ) ] );
```

```
fwrite( fid , [ sprintf('\n') ] );
end
if ~isreal( dominantEigenvaluesMatrix( I ) ),
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ 'Effect on the Frequency:' ] );
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [ sprintf('\n') ] );
for K = 1:size( independentCyclesVerticesMatrix , 1 ) ,
fwrite( fid , [ 'Loop ' num2str( IX_I( I , K ) ) ' (Polarity: '
num2str(
signIndependentCyclesMatrix( I , IX_I( I , K ) ) ) '): ' num2str(
dummy_IndependentCy
clesElasticity_Sorted_I( I , K ) ) ] );
fwrite( fid , [ sprintf('\n') ] );
end
end
end
fwrite( fid , [ sprintf('\n') ] );
fwrite( fid , [
'______________________________________________________' ] );
fwrite( fid , [ sprintf('\n') ] );

fclose( fid );
```

Appendix F

External Functions

The following functions are parts of the Digraph toolbox for Matlab 4.2, with minor changes to suit Matlab 6.5.

The digraph toolbox is available at:

http://fuzzy.iau.dtu.dk/download/digraph.zip

F.1 reachabi.m

For all nodes, it tests if one node is reachable from another.

```
function r = reachabi(m) ;
% Reachability matrix of m
%
% function r = reachabi(m) ;
%
% m square matrix
% r square boolean reachability matrix
%
% r(i,j) = 1 if node i is reachable from
% node j; 0 otherwise.
% The matrix m is
% turned into a boolean inside.

r = m & 1;
r = r | eye(size(r));
aux = zeros(size(r)) ;
while prod(prod(+(aux == r))) == 0,
aux = r ;
r = r | (+r * +r) ;
end ;
```

F.2 delzrow.m

In a matrix, it removes any row that all its elements are zeros.

```
function res=delzrow(a)
%Remove zero rows in a matrix
%
%function res=delzrow(a)
%
res=a;
if isempty(a)
return;
```

```
end

if size(a,2)>1,
res = a(find(sum(a.')),:);
else
res=a(find(a~=0));
end;
```

F.3 allpathn.m

Find all paths between two nodes.

```
function allpaths = allpathn(from,to,a)
% All paths between two nodes
%
% function allpaths = allpathn(from,to,a)
%
% from from-node, a number
% to to-node, a number
% a square successor matrix (n,n)
%
% allpaths matrix (?,n), each row contains the node numbers in a walk on a
% path; the matrix is padded with 0's to the right
%
% The algorithm is a traversal of the digraph. It uses the reachability matrix
% to prune the traversal.

a=a&1;
r=reachabi(a);
[n,dum] = size(a) ;
paths = from ;
allpaths = [] ;
emptypath = zeros(1,n+1) ;
while ~isempty(paths),
[maxp,dist] = size(paths) ;
newpaths = [] ;
for i=1:maxp,
curpath = paths(i,:) ;
candids = find(a(:,curpath(dist))&r(to,:)') ;
for j = 1:length(candids),
cand = candids(j) ;
p1 = all(cand ~= curpath) ;
p2 = cand == to ;
if p2,
newpath = emptypath ;
newpath(1:(length(curpath)+1)) = [curpath,cand] ;
allpaths = [allpaths;newpath];
[p,dum]=size(allpaths);
```

```
% disp([int2str(p),' paths found']);
elseif p1 ,
newpath = [curpath,cand] ;
newpaths = [newpaths;newpath] ;
end ;
end ;
end ;
paths = newpaths ;
end ;
```

F.4 allcycsn.m

All cycles in graph of a matrix including the node indexes.

```
function cycles = allcycsn(a)
% All cycles in graph of matrix A
%
% function cycles = allcycsn(a)
%
% a square (boolean) successor matrix (n,n)
% cycles' matrix (?,n), each row contains the node numbers in a walk around
% a cycle; the matrix is padded with 0's on the right
%
% The algorithm is an exhaustive traversal of the digraph with pruning. An
% early version is in APL in Evans & Larsen (1981). To get the cyc-
% les as boolean lists, use 'allcycs'.

[n,dum] = size(a) ;
paths = (1:n)' ;
cycles = [] ;
emptycyc = zeros(1,(n+1)) ;
while ~isempty(paths),
[maxp,dist] = size(paths) ;
newpaths = [] ;
for i=1:maxp,
curpath = paths(i,:) ;
candids = find(a(:,curpath(dist))) ;
for j = 1:length(candids),
cand = candids(j) ;
p1 = all(cand ~= curpath) ;
p2 = cand == curpath(1) ;
p3 = cand > curpath(1) ;
if p2,
newcyc = emptycyc ;
newcyc(1:(length(curpath)+1)) = [curpath,cand] ;
cycles = [cycles;newcyc];
elseif p1 & p3,
newpath = [curpath,cand] ;
```

```
newpaths = [newpaths;newpath] ;
end ;
end ;
end ;
paths = newpaths ;
end ;
```