The Report Committee for Marihebert Josefina Leal Vasquez
Certifies that this is the approved version of the following report:

Design and Implementation of a Software Framework to Model and Simulate Engineering Systems using BondGraphs.

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor:

_____

Kathleen S. Barber

Co-supervisor:

_____

Benito R. Fernández

Design and Implementation of a Software Framework to Model and Simulate Engineering Systems using Bond Graphs.

by

Marihebert Josefina Leal Vasquez, B.E.;M.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2015

Dedication

To my son

Acknowledgments

I would like to thank , Dr Benito Fernandez and Dr Suzanne Barber for providing guidance and for allocating their valuable time to this endeavor. I wholeheartedly thank my husband, family and friends for their encouragement and support. Finally, I am grateful to CLEE for providing the opportunity to accomplish my academic aspirations.

Abstract

Design and Implementation of a Software Framework to Model and Simulate Engineering Systems using BondGraphs

by

Marihebert Josefina Leal Vasquez, M.S.E.
The University of Texas at Austin, 2015

Supervisor: Kathleen S. Barber
Co-supervisor: Benito R. Fernández

This report presents the development of a software framework for deriving explicit state equations in symbolic form of physical systems described by bond graphs.

This program called **Bond Graph Tool** is an open-source object-oriented implementation in Python, using the Tkinker and SymPy libraries. The Tkinker library has several functions that enables the user to command operations and display the results. SymPy is a Python library for symbolic mathematics, which permits the state-equations derived from the Bond

Graphs in symbolic form.

The Bond Graph Tool provides a graphic interface for drawing and editing Bond Graphs. The program allows to automatically assign the causalities on the Bond Graph. Output from the program is in the form of symbolic equations.

The program handles the basic 1-port and 2-port elements as well as multiple ports junctions and derivative causality. The current version of the program, however, has limitations in handling several difficult features in bond graph.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

Engineering often entails the design, analysis, or control of dynamic systems. System Dynamics draws on a variety of engineering specialties to form a unified approach to study dynamic systems [2].

The discipline of System Dynamics focuses on the synthesis of mathematical models to represent dynamics responses of physical systems for the purpose of analysis, design and/or control. Any attempt to design a system must often requires a prediction of its performance before a prototype of the system itself can be designed in detail or actually built. Such prediction can be made by a mathematical description of the system's dynamic characteristics.

This report will utilize one of the techniques for generating mathematical models called the bond graph method, introduced in the late 1950's by Henry M. Paynter [1]. This approach allows one to study the structure or interconnection of a system model, which is a direct reflection of the physical system. The nature of parts of the model and the manner in which the parts interact is communicated via a graphical format. Analogies between various types of systems are made evident, and experience in one field can be extended to other fields [2].

Bond Graphs have several distinctive advantages over other diagram techniques.Bond graphs can account for energy transfer between different energy domains. Second, Bond Graphs can clearly depict multiple inputs and outputs. Third, bond graph elements support non linear constitutive relationships as well as elements which can store or dissipate energy from multiple energy domains simultaneously.

One of the most important aspect of the Bond Graph method is that it provides a convenient link between a real physical system and an abstract functional model [2]. A Bond Graph provides this link because once it is drawn, and the constitutive relationships for each element are obtained, a systematic and routine method can be used to convert the graph into a desired mathematical formulation, for example ordinary differential equations.

Even though the conversion routine from a bond graph to a state equation form is structured, human effort hits limitation as the number of component increases and links between the components become complex. Several attempts to automate this task have been. Enport [5] by Ronald Rosenberg Development, Modelica [7],CAMPG [6], 20-Sim [8], MTT [9] and OpenModelica [10]are some examples of this effort.

A program, Bond Graph tool, was developed in Python, which is an open source program. It provides a graphic interface for drawing and editing bond graphs and computes the state representation of the dynamic system modeled from the bond graph in a symbolic form.

1.1 Overview of Program

The program was developed in Python using the Tkinker and SymPy libraries. Tkinter is the standard Python interface for developing graphical user interface and SymPy is a Python library for symbolic mathematics, which let you express the state equations (derived from the bond graphs) in symbolic form.

The current version of the bond graph software framework handles some of nonbasic bond graph structures. The program is also designed to support a certain number of algebraically determined derivative causalities.

1.2 Organization of the Report

This report presents the development of a Bond Graph software framework for calculating mathematical system models represented using Bond graphs.The rest of the report is organized as follows. Chapter 2 contains a brief description of the general Bond Graph element, along with the constitutive relationships which are defined according to the causality information for every elements. Chapter 3 describes the overall structure of the program. Chapter 4 presents illustrative examples of bond graph models and their analysis using the program developed. Chapter 5 gives conclusions and recommends future work.

Chapter 2 About BondGraph

This chapter shows essential background information on Bond Graphs. It briefly introduces the Bond Graphs elements, the junctions and their functional relationships which are determined by constitutive relation and causality information. Also, how equations are derived.

2.1 Bond Graph Modeling

The Bond Graph notation often allows one to visualize aspects of the system more easily than would be possible with just the mathematical form or with some other graphical notation specially designed for a single energy domain or for a signal flow rather than the power flow [2]. Using bond graphs and the classification of power and energy variables presented, it turns out that only a few basic types of multiports elements are required in order to represent models in a variety of energy domains.

In order to developed state determined functional relationship for systems and subsystems, a set of bond graphs modeling elements are needed which have precise functional descriptions[3]. These essential modeling atoms can either:

1. Store energy

2. Transform energy

3. Dissipate energy

4. Join or add sub-systems to form total systems

5. And in order to describe the effects between the environment and the system, be a source of energy.

State determined equations for the bond graph models can be formulated by using the precise functional relationships represented by bond graph elements [3].

2.2 Energy and Power Variables

In various energy domains, there are variables that when multiplied together give power. For example, force and velocity when multiplied give power (i.e.,$P(t) = F(t)v(t)$), and voltage and current also multiply to give power (i.e.,$P(t) = e(t)i(t)$). Power is generally defined as the multiplication of and *effort* (a force-like variable), $e(t)$, and *flow* (a velocity-like variable), $f(t)$,

$$P(t) = e(t)f(t) \tag{2.1}$$

Effort and flow can be related to the generalized energy variables *momentum*, $p(t)$ and *displacement*, $q(t)$. The generalized momentum is defined as the integral of the effort

$$p(t) \equiv \int e(t)dt, \tag{2.2}$$

which means then that the effort is the time rate of change of the momentum,

$$e(t) = \frac{dp}{dt} = \dot{p}. \tag{2.3}$$

The generalized displacement is defined as the integral of the flow,

$$q(t) \equiv \int f(t)dt, \tag{2.4}$$

In bond graphs, elements are connected by *bonds* through *power ports*. Each bond represents and effort-flow pair that when multiplied give the power entering or leaving the attached ports.

2.3 Basic 1- port Elements

The constitutive relationships for basic 1-port elements relating the energy and power variables identify whether that element stores potential energy (C-element), stores kinetic energy (I-element), or dissipate energy (R- element).

2.3.1 R-Elements

R- elements are basic, 1-port elements that dissipate energy. The analogy of electric resistor, R applies to explain the dissipative element in bond

graph. Energy dissipating elements do not specify effort or flow types because one variable can be expressed in terms of either variable.

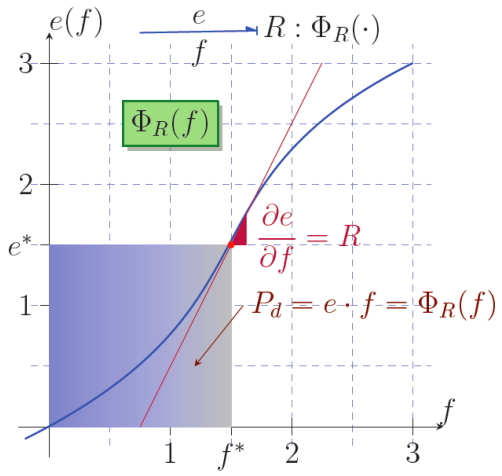

Figure 2.1: BondGraph characteristics of a Resistor (R)

Therefore, the function may be read two ways: the effort can be found the flow is given, or the flow can be found when the effort is given as shown in equation 2.5.

$$e = \Phi_R(f) \quad or \quad f = \Phi_R^{-1}(e) \tag{2.5}$$

where $\Phi_R(f)$ and $\Phi_R^{-1}(e)$ are in general nonlinear functions.

2.3.2 C-Elements

C-elements are basic elements that store potential energy. They are characterized by a constitutive relationship that directly relates effort to generalized displacement,

$$q = \Phi_C(e) \quad or \quad e = \Phi_C^{-1}(q) \tag{2.6}$$

Note that in 2.6, generally either displacement is a nonlinear function of effort, or effort is a nonlinear inverse function of displacement.

Since the energy is stored in potential form by integration in terms of displacement, C, capacitor is used as a bond graph symbol to denote the constitutive relation.

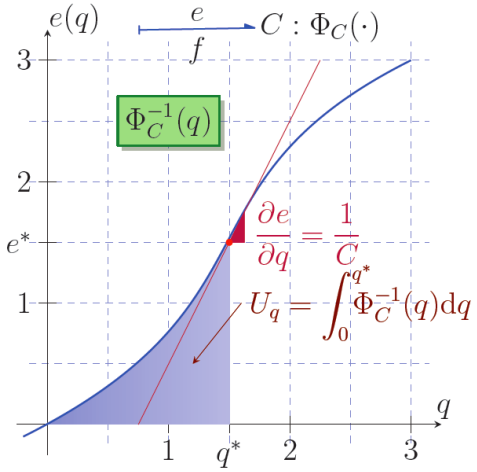

Figure 2.2: BondGraph characteristics of a Capacitor (C)

2.3.3 I-Elements

I-elements are basic elements that store kinetic energy. This constitutive relation is applied to model inductance effect in electrical systems and mass or inertia effect in mechanical or fluid system, where the bond graph symbol I, inertia originates from. The constitutive relationship that directly relates momentum to flow is,

$$p = \Phi_I(f) \quad or \quad f = \Phi_I^{-1}(p) \tag{2.7}$$

Note that the equation above, generally either momentum is a nonlinear function to flow, or flow is a nonlinear inverse function of momentum.

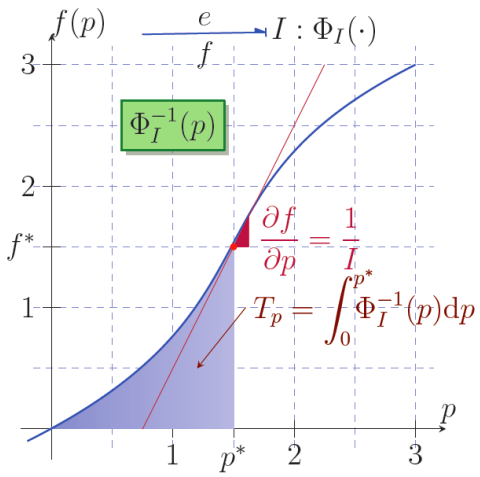

Figure 2.3: BondGraph characteristics of an Inductor(I)

2.3.4 Effort and Flow Sources

Source elements are thought of as supplying power to the system such as voltage or current sources, vibration shakers, external forces, pressure sources, etc. The system inputs can be either effort source or flow source, where the type of source defines the variable controlled by the source, which, for an ideal source, is independent of the co-variable which is defined by the system which the source supplies. Since a source maintains one of the power variables constant or a specified function of time, it can supply an in definitely large amount of power to the system. Bond graph notation uses E

as an effort source F, as a flow source.

2.4 Basic 2-port Elements

Transforming notations are required to interpret the system that the variables on the output port is scaled to input variables. There are elements that neither dissipate nor store energy but do transmit energy from one element or junction to another while often times interfacing between various energy domains. Bond graph introduces two types of transforming elements depending on the relationship between the input variables and output variables: transformer (TF) and gyrator (GY).

2.4.1 Transformers (TF Elements)

Transformers, designated by TF, are power conserving although the effort on the output port is scaled by the transformer ratio to the effort on the input port.

Equation 2.8, for example, is the case of transformation that the effort of the output port and the flow of the input port are formed from the other side. $\Phi_T(x)$ denotes the transformer modulus as a function of a signal state x.

$$e_2 = \Phi_T(x).e_1 \quad ; \quad f_1 = \Phi_T(x).f_2 \tag{2.8}$$

2.4.2 Gyrators (GY Elements)

Another way to satisfy the power balance between input and output is embodied in the gyrator, which is symbolized as GY. The parameter,

modulus, governs the ratio of input and output in both cases.

The constitutive relationships for a gyrator are present in equation 2.9. It also shows that modulus can be changed without changing the fact that the power is conserved in two ports.

$$f_1 = \Phi_G(x).e_2 \quad ; \quad f_2 = \Phi_G(x).e_1 \tag{2.9}$$

2.5 Junction Elements

Junction elements serve to interconnect other Bond Graph elements into subsystem or system models. The idea is to represent in multiports form the two types of connections that either effort (or flow) is fixed and the co-variables must sum to zero. This notation shares the principle with Kirchoff's Laws of Current and Voltage. Thus, common effort junction, which is conveniently termed '0' is analogous to parallel connection in circuit and common flow or '1' junction to series connection. Usage of these junctions in bond graph is depicted in the following diagram.

0 junction of figure 2.4, the causality stroke determines that only k bond which has input effort causality provides the effort information in the junction. Flow of this bond is the sum of the flows of the rest of bonds. Similarly, the k bond with flow specified stroke carries the flow into the 1-junction and obtains its effort from the other bonds which are effort specified. Table 2.1 summarizes the primary and secondary conditions and the sign convention for 0 and 1 junctions.

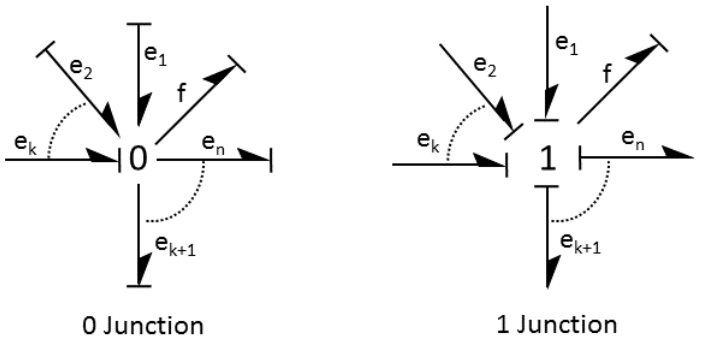

Figure 2.4: Common Effort Junction and Common Flow 1-Junction

Table 2.1: Summary of junction conditions

Junction Type	**Primary Condition**	**Secondary Condition**	**Bond Direction Indicates**
0-Junction	common effort, $e_1 = e_2 = ... = e$	sum of flows, $\sum f_i = 0$	Flow sign convention (sign of the flows)
1-Junction	common flow, $f_1 = f_2 = ... = f$	sum of efforts, $\sum e_i = 0$	effort sign sign convention (sign of efforts)

2.6 Derivation of State Equations

State equations can be derived by starting with the bonds attached to the independent energy storing elements and solving for the rate of the momenta of the independent I elements and the rate of the displacements of the independent C elements. These are the time derivatives of the state variables [4].

The general algorithm for the state equations is simply to express all the elemental functional relationships in causal form and then sort these relations such that they can be evaluated in an explicit sequential form.

Although, there is no established process for deriving differential equations from bond graphs, the following is provided as a set of guidelines [3]:

1. Assign causality
2. Label efforts and flows on the energy- storing elements
3. Apply the primary conditions
4. Apply the secondary conditions

This section presents the general idea of formulating state equations from bond graph with complete explicit causality and derivative causality. The concept introduced in this section is used to develop a systematic method to derive state equations in the Bond Graph Interface Tool.

2.6.1 Complete Explicit causality

Each bond has effort and flow information. The matrix representation is useful for describing the data structure of such an information. An example of the matrix generated from a simple bond graph of the figure 2.5 is shown in equation 2.15.

The causality assigned on the bond graph above specifies what type of information each bond carries into the system. The constitutive relations that correspond to the causality and the element type of the bond are stores in the following equations,

$$S_e \begin{cases} e_1 = E(t) \\ f_1 = f_2 \end{cases} \tag{2.10}$$

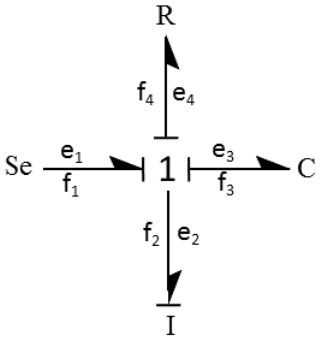

Figure 2.5: Bond Graph with Explicit Causality

$$I \begin{cases} \dot{p_1} = e_2 \\ f_2 = p/I \end{cases} \tag{2.11}$$

$$C \begin{cases} e_3 = q/c \\ \dot{q} = f_3 \end{cases} \tag{2.12}$$

$$R \begin{cases} e_4 = R.f_4 \\ f_4 = f_2 \end{cases} \tag{2.13}$$

and the equation 2.14 shows the conditions for the junction 1

$$\begin{aligned} & f = f_1 = f_2 = f_3 = f_4 \\ & \textstyle\sum e_i = 0 \\ & e_2 = E(t) - e_3 - e_4 \end{aligned} \tag{2.14}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1/I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1/C & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -R & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} e_1 \\ f_1 \\ e_2 \\ f_2 \\ e_3 \\ f_3 \\ e_4 \\ f_4 \\ E \\ p \\ q \end{pmatrix} = \begin{pmatrix} 0 \\ \dot{p} \\ 0 \\ \dot{q} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.15}$$

The Bond Graph in figure 2.4 informs that the flow of bond 1,4, and 3 are carried by bond 2 and effort of bond 2 is KLC based sum of the rest of bonds. Matrix 2.15 stores the constitutive relations of each bond and junction information of the bonds. The relation in matrix 2.15 have to be satisfied simultaneously at each instant of time in order to obtain a solution to the system. These equations can be satisfied in a sequential manner by starting with allocating time derivatives of state variables.

State variables are independent energy storing elements of the system. They are easily visible from the causality of the bond graph. Initial state

equations consist of the rates of the state variables.

$$\begin{aligned} \dot{P}_4 &= e_4 \\ \dot{q_2} &= f_2 \end{aligned} \tag{2.16}$$

Generating state equations proceeds in substituting the corresponding elements of the matrix 2.15 to the equation 2.16. Arbitrarily, starting from $\dot{P}_4$ in the initial state equation 2.16, it can be seen from the matrix 2.16 that $\dot{P}_4$ is a function of e_4 and the effort e_4 is a function of e_1,e_2, and e_3. Temporarily pushing e_2 and e_3 onto an auxiliary stack and continuing with e_1 it can also be seen from the matrix 2.15that e_1 is solely function of time t which needs no further substitution. Popping e_2 off the auxiliary stack, this effort is a function of q_2, which also needs no further substitution and continues in the same manner for the rest of variables.

Equation 2.17 shows an intermediate form of the state equations after replacing with the constitutive equations in order to find an explicit expression for $\dot{P}_4$.

$$\begin{aligned} \dot{P}_4 &= E(t) - q/c - R.f_3 \\ \dot{q_2} &= q \end{aligned} \tag{2.17}$$

The substituting procedure continues until all the variables become the state variables. The one of the intermediate variables in the equations 2.17 is f_3, which is equal to f_4. This variable can be finally replaced with the constitutive relation of f_4,

If there exists no intermediate variables, the substituting process finishes.Finally, equation 2.18, explicit form of the state equations is formulated.

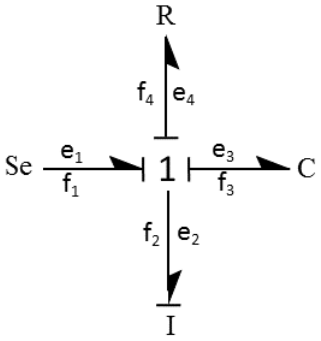

Figure 2.6: Bond Graph with Derivative causality

Sequential state equation generation is completed since all the rate relations have been generated.

$$\begin{aligned} &-Rf(t) - \tfrac{q(t)}{C} + E(t) - \tfrac{\partial}{dt}p_1(t) = 0 \\ &\tfrac{\partial}{dt}q(t) - \tfrac{p_1(t)}{I} = 0 \end{aligned} \tag{2.18}$$

2.6.2 Derivative Causality

Derivative causality occurs when either a flow is imposed on an I or an effort on a C when propagation of causality after assignment of a source or another independent energy storing element. The result is a dependent energy-storing element. The result is a dependent energy-storing element. Figure 2.6 shows that I element of bond 2 has derivative causality.

The equation generation starts just as in the previous case with producing constitutive relation and junction matrix. The bond of derivative causality provides momentum information which is a function of flow vari-

able into the system. The effort constitutive relationship of this bond becomes time derivative of the momentum information as shown in equation 2.19. The d/dt label stands for derivative or dependent state. During substitution the d/dt variable is ignored and is only included in the generation for completeness.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1/C & 0 & 0 & 0 \\ 1 & -1/C & 0 & 0 & 0 & 0 \\ R & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & I_4/I_2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} f \\ q \\ p \\ e_2 \\ e_3 \\ e_4 \end{pmatrix} = \begin{pmatrix} \dot{p} \\ 0 \\ 0 \\ 0 \\ 0 \\ E(t) \end{pmatrix} \tag{2.19}$$

The procedure continues with determining a state variable and replacing the intermediate variables with the corresponding variables of the matrix 2.19.

The state equation is now expressed only in terms of the state variables, P_2, as shown in equation 2.20. A reduction to a completely explicit form of the state equation 2.20 requires differentiation of time derivative term in the left hand side of the equation and solve for time derivative of the state variable in order to express the equation in terms of the state variable.

$$\begin{aligned} \dot{P}_2 &= E(t) - e_3 - e_4 \\ \dot{P}_2 &= E(t) - Rf_3 - I_1/I_0 * p_2 \end{aligned} \tag{2.20}$$

Chapter 3 Framework Design

This chapter presents a discussion of the Bond Graph tool. The program was developed as an open source object oriented in Python. The program is structured with mainly three parts: create the bond graph adding all elements needed, connect all elements, and finally calculate the state equations in a symbolic form.

3.1 Motivation

The bond Graph concept [2] proved to be highly successful for teaching dynamic system modeling has motivated the development of different tools for modeling dynamic systems using bond graphs. Enport [5] by Ronald Rosenberg Development, which was one the first attempts is no longer available. Others, like Modelica [7], CAMPG [6] and 20-sim [8] are commercial software.Just, few open source tools has been implemented, like MTT [9] and OpenModelica [10] . The main objective of this development is the implementation of an open source object oriented software, which can be available to anyone. The software will allow enhancements or modifications by anyone.

3.2 Approach

Bond Graph Tool is an open-source object-oriented software framework developed in Python. Object oriented functions let us get rid of duplicate code and organize the code to make changes easier. Python combines remarkable power with very clean syntax, and it is portable across windows, Linux and Mac with very little modifications. Also, Python has interfaces to many systems call and libraries. For this development two libraries were used: tkinker and SymPy. Tkinter is the Python library for developing graphical user interfaces(Tk GUI) [11] [12]. SymPy is a Python library for symbolic mathematics, which will allow the state equations derived from the Bond graphs be created in symbolic form . Beyond use as an interactive tool, SymPy can be embedded in other applications and extended with custom functions.

The BondGraph Tool (BGT) offer the following main functions:

- provides a graphic interface for drawing and editing of Bond graphs.
- performs automatic causality assignment to the bond graph.
- computes the state space representation of the dynamic system modeled from the bond graph in symbolic form.

BGT's GUI has a window with three buttons: Bond Elements, Connect, and calculate equations, that are described in the following subsections.

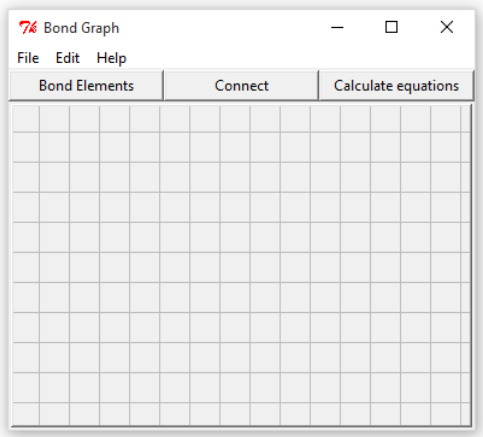

Figure 3.1: Bond Graph tool main window

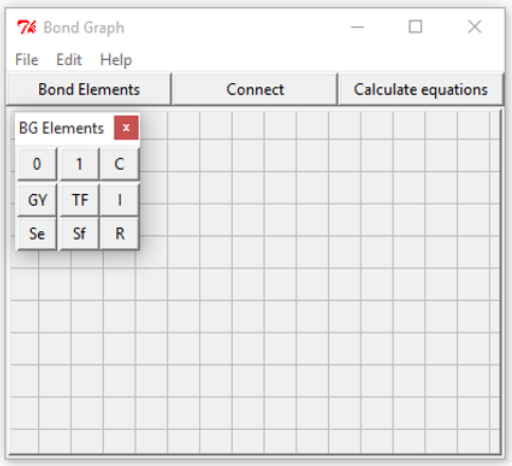

Figure 3.2: Bond Graph tool Bond Elements window

3.2.1 Bond Elements

After "clicking" this button with the cursor a pop-up window appears with all basic 1-port elements (R, I, C), junctions (0,1), the basic 2-port elements (TF,GY), and sources (Se, Sf). See 3.2.

After once the Bond Elements window is opened, drag and drop any element needed to draw the specific Bond graph. After all elements desired

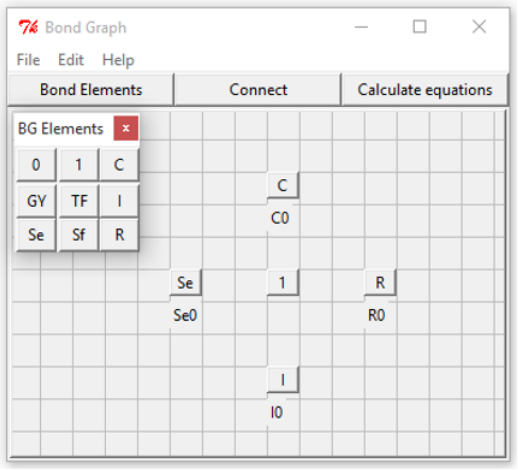

Figure 3.3: Bond Graph tool Bond Elements

are on the workspace window, all of them have to be adjust to the grid, as show in figure 3.3.

3.2.2 Connect

The Connect button lets the user connect any two element to build the Bond graph. When this button is activate, all elements on the workspace window are blocked. If an element is need to move is necessary to click The bond Elements button again.

After clicking the Connect button, each element is connected starting with a “click” on the element where the bond is to begin.Then, click” at the point where the bond is to end. The bond will be drawn with a half arrow at its head indicating the positive direction of the power. These steps are repeated until the entire bond graph structure is completed. If an error occurs selecting the first element, it is possible to disable the connection by clicking on the workspace window without selecting any item.

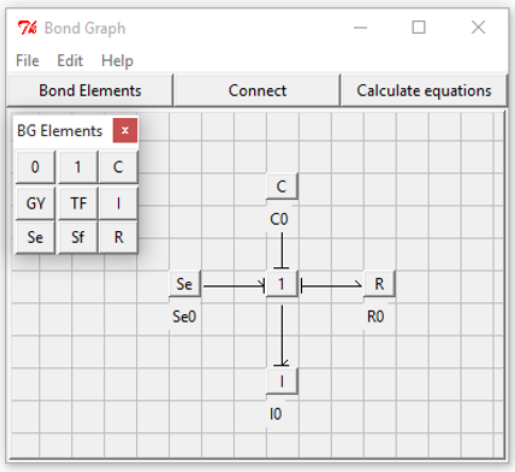

Figure 3.4: Bond Graph drawing

$$
\begin{aligned}
&\text{================ to i= 1 ================}\\
&-\frac{d}{dt}(q_2(t)) + \frac{p_1(t)}{I_0}\\
&\text{================ to i= 2 ================}\\
&E(t) - \frac{d}{dt}(p_1(t)) - \frac{R_0 \cdot p_1(t)}{I_0} - \frac{q_2(t)}{C_0}
\end{aligned}
$$

Figure 3.5: State equations

The causalities are assigned automatically whenever is possible. The algorithm implemented in the Bondgraph tool for assigning causality follows the procedure described in [2].

3.2.3 Calculate Equations

After the calculate equations button is clicked, the tool generates the state equations in symbolic form. The algorithm checks to see if all causalities have been assigned. If not, it ask the user to do the assigment manually.

3.3 Coding

The Bond Graph Tool is comprised of four classes and fourteen subroutines, which are explained below. The complete program is shown in the Appendix.

Class Dragged: It creates the correspondent objects to the Bond-Graph elements, the variables are defined, like as type, name, XY coordinates and the symbolic values of effort and flow. This class has the following functions or subroutines:

- `Dnd_end()`:It allows the drag and drop of the elements from the palette to the work space window, creating the BG element (object) and assigning the defined values.

- `Appear()`: Shows the elements (objects) in the work space window, drawing a button and a label with the assigned names.

- `Move()`: Allows that the elements to be moved in the work space windows.

- `Press()`: Contains the connection routines, namely, to draw the bond (line and the half arrow) and the automated assignment of causalities. This function is enabled when the "Connectâ button is pressed, and is disabled when the "Bond Elementsâ button is pressed.

Class CanvasDnD: This class handles the tools to use in the canvas that serves as the work space window. This class has the following functions or subroutines:

- `Dnd_enter()`: This function gets the dragged item using the *Appear()* function, which was defined above, and it creates an object where the dragged object properties are copied.

- `Dnd_motion()`: Captures XY coordinates when the object is dragged in the work space window, and draws the object on the new position using the *Move()* routine already defined.

Class TrashBin: This class allows the deletion of the elements in the work space window. This class is not working in this version.

Class PaletElemBG: This class creates the palette window containing the different Bondgraph elements to be used.

- `Main()`: It is the main routine, which is started when the program runs. It contains the calls to the others routines, also it creates the buttons "Bond Elements", "Connect" and "Calculate equations" and link them with the appropriate routines.

- `Main()`: It is the main routine, which is started when the program runs. It contains the calls to the others routines, also it creates the buttons "Bond Elements", "Connect" and "Calculate equations" and link them with the appropriate routines.

- `New_Window()`: Open the palette window, and define two global variables to deactivate the move routine in the work window.

- `CreateLine()`: Change the value on the variable to enable the Press routine.

- `CalcEq()`: Verifies if all elements have causality, if some element have not causality *AsigCausal()* is called, otherwise search the first source element, calls *CalPartial()* and prints the obtained equations.
- `AsigCausal()`: Assigns the causality in a manual way.
- `CalPartial(numElem, BG_Ini, BG_junc, Index)`: Calculates the equations.
- `MenuCreator()`: Creates the main menu. In this version this option is disabled.
- `GridCanvas()`: Draws the grid in the work space window.

The development of the BondGraph tool include 4 fundamental functions:

- Elements Built
- Elements Connection
- Causality assign
- Calculation of equations

3.3.1 Element Built

The Bond graph elements from the palette, which appear after clicking the Bond Elements button, are objects in the program. Each bond graph element (C, R, I, etc) is an object with a name variable, which assign a unique number (ID), which is incremented when an element of each type is

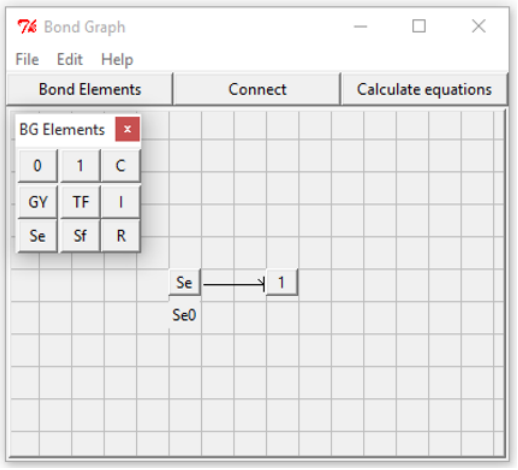

Figure 3.6: Bond Graph tool Bond Elements

added. For example, when an I element is added, the name I0 is assigned, and if another I element is added, for the new one I1 name is assigned. Also, each object has the (x,y) position on the workspace window, and the relation of effort and flow of each Bond Graph element.

3.3.2 Elements Connection

In order to connect to elements, three steps are performed:

First, all elements connected are blocked even if the "Bond Elements" is pressed again. So an array containing the information of each pair of connected elements is created. For example, if an "Se" element is connected to "1" element, as are shown in the Fig 3.6 , the pair [Se0, 10] is created. Finally, the function for assigning causality is called. This function is explained in the following subsection.

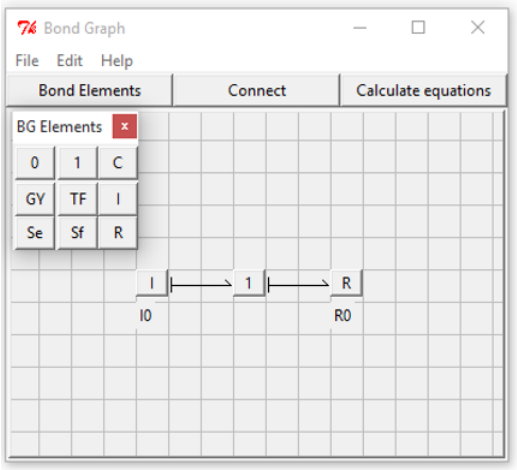

Figure 3.7: Bond Graph tool Bond Elements

3.3.3 Causality assign

When two elements are connected, the function to assign causalities is activated, which asks if one of the elements is a junction "0" or "1", then asks whether one of the connected elements is a Se, Sf, I or C. In case that one of the elements is Se, Sf, I or C, their preferred causality is assigned. For example, if an I element is connected with a "1" junction , the causality will be 1 ⇀ | I and a flag is raised indicating that the rest of the elements connected to this junction have the causality of outgoing effort. Also, if an R element is connected to the same 1-junction, the causality will be 1 ⇀ | R, as it is shown in 3.7. For the case where two R are connected to the same junction, these elements have not preferred causalities, which means R0 ⇀ 1 ⇀ R1. In this case, the causality is assigned manually when the "Equation Calculate" button is pressed.

This function also creates an array containing the assignment information of the causalities, for example if the connected elements are 1 ⇀ |

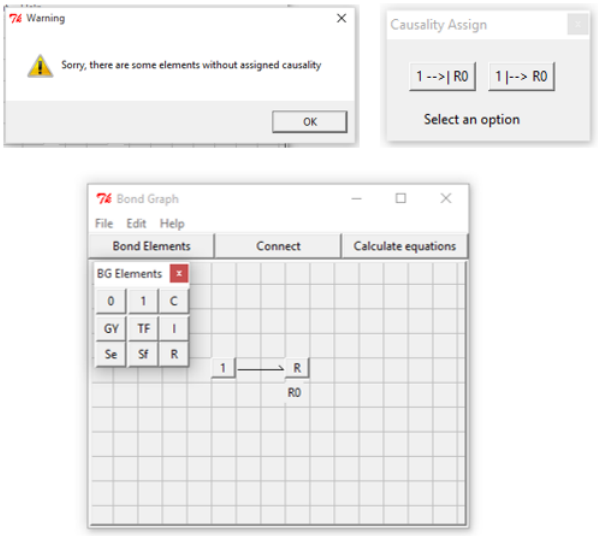

Figure 3.8: Bond Graph tool Bond Elements

I, the pair created will be [0, 1], where the 1 value indicates which element contains the causality, in this example, the "I" who has the causality stroke.

3.3.4 Calculation of equations

When the "Calculate Equation" button is clicked, the first thing that the program does is check whether all elements have their causality assigned. if not, it proceeds to assign causalities, to the elements that do not have it, in a manual way, as shown in 3.8 . After that, the "Calculate Equation" button should be pressed again. After all causalities are assigned, an array filled of zeros, named *EqIn*, is created filled with the size of the number of elements in the bond Graph. Then, proceed the calculations of the equations using the functions *CalEq()* and *CalPartial()* follow.

The function *CalEq()* first locates a source, either effort or flow, then locates to which junction is connected and finally, calls the *CalPartial()* func-

tion.

Then, the function *CalPartial()* locates all elements connected to this junction. All the relations of efforts and flows are assigned and are added in two variables, *sumaE* for the efforts and *sumaF* for the flows. Also this function let us know how many equations are necessary, asking whether the element connected to the junction is an "I" or "C" and if they have their preferred causality. If the answer is yes, this element is marked by 1 in the corresponding element in the array *EqIn*. If the element is connected to another junction, "0" or "1", the same routine is called recursively. If the connected component is a transformer or gyrator another routine is called, which calls the *CalPartial()* routine recursively, but the outputs given by these function are handled in order to satisfy the relations of those elements, for example when working with a given transformer, $\{e_1, f_1\} \rightharpoonup TF : m \rightharpoonup \{e_1, f_2\}$, should be satisfied that $e_1 = m * e_1$ and $f_2 = m * f_1$.

Chapter 4 Application Examples

This chapter illustrates how the Bond Graph tool can be used to analyze (in an easy way) the dynamic behavior of some typical systems. The consecutive sections show three physical examples to demonstrate the ideas and techniques presented in the previous chapters. The first example is a system of two masses linked with two springs. The second example is a RLC circuit. The last example is a basic hydraulic system.

4.1 Mechanical System

The physical system show in figure 4.1 is a linear mechanical system, where each velocity is associated with a 1-junction, including a reference (inertia) one. Masses are linked as I-elements to the corresponding 1-junction. Springs and dissipate elements are linked to 0- junctions connecting appropriate 1- junctions. The rest of the elements are inserted and power orientation are choose.

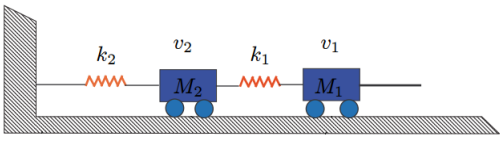

Figure 4.1: Mechanical System

The bond graph of the system is shown in figure 4.2

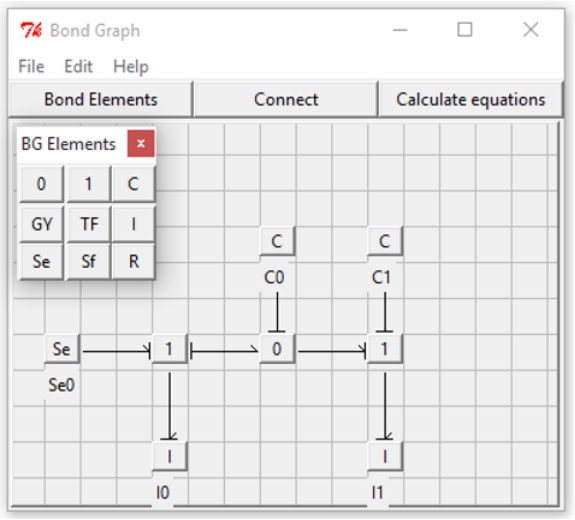

Figure 4.2: Bond graph of the Mechanical System

The state equations in symbolic form generated by the Bond Graph tool are shown in figure 4.3

$$
\begin{aligned}
&\text{to i= 1} \\
&E(t) - \frac{d}{dt}(p_1(t)) - \frac{q_3(t)}{C_0} \\
&\text{to i= 3} \\
&-\frac{d}{dt}(q_3(t)) - \frac{p_5(t)}{I_1} + \frac{p_1(t)}{I_0} \\
&\text{to i= 5} \\
&\frac{d}{dt}(p_5(t)) - \frac{q_6(t)}{C_1} + \frac{q_3(t)}{C_0} \\
&\text{to i= 6} \\
&-\frac{d}{dt}(q_6(t)) + \frac{p_5(t)}{I_1}
\end{aligned}
$$

Figure 4.3: State equations of the Mechanical System

4.2 Electrical System

In this example, an RLC circuit is modeled. An RLC circuit (or LCR circuit) is an electrical circuit consisting of a resistor, an inductor, and a

capacitor. The RLC part of the name is due to those letters being the usual electrical symbols for resistance, inductance and capacitance respectively. The RLC circuit is given in the figure 4.4.

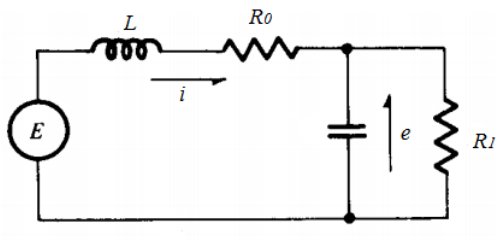

Figure 4.4: Electrical System

In electrical networks, the port variables of the bond graph elements are the electrical voltage over the element's ports and electrical current through the element's ports. Note that a port is an interface of an element to other elements; it is the connection point of the bonds. The power being exchanged by a port with the rest of the system is the product of voltage and current.

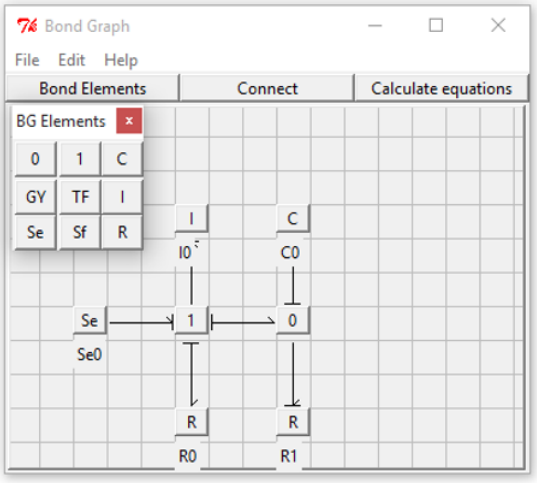

Figure 4.5: Electrical System Bond Graph

The voltage is mapped onto the domainindependent effort variable

and the current maps onto the domainindependent flow variable (the current always on the side of the arrow). The 1-junction means that the current (flow) through all connected bonds is the same, and that the voltages (efforts) sum to zero, considering the sign. The sign is related to the power direction (i.e.direction of the half arrow) of the bond. The summing equation is the Kirchhoff voltage law. Parallel connections, in which the voltage over all connected elements is the same, the bondgraph mnemonic is a 0âjunction. A 0âjunction means that the voltage (effort) over all connected bonds is the same, and that the currents (flows) sum to zero, considering the sign. This summing equation is the Kirchhoff's current law. The bond graph of the system is shown in figure 4.5

The state equations in symbolic form generated by the Bond Graph tool are shown in figure 4.6

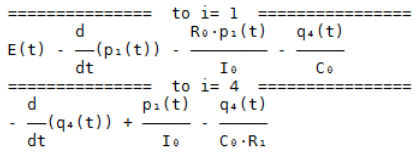

Figure 4.6: State equations of The Electrical System

4.3 Hydraulic System

Finally, a basic Hydraulic system is examined (see Fig 4.7. Hydraulic circuits are very similar to electric circuits. The bond graph of the system is shown in Fig 4.8

The state equations in symbolic form generated by the Bond Graph tool are shown in Fig 4.9.

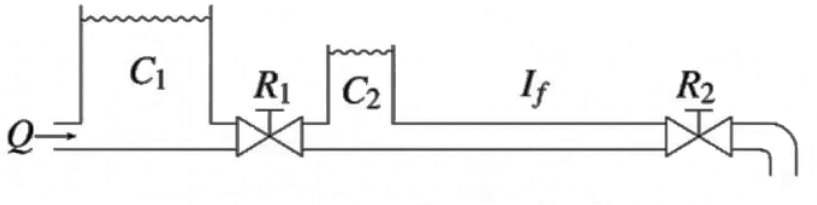

Figure 4.7: Hydraulic System

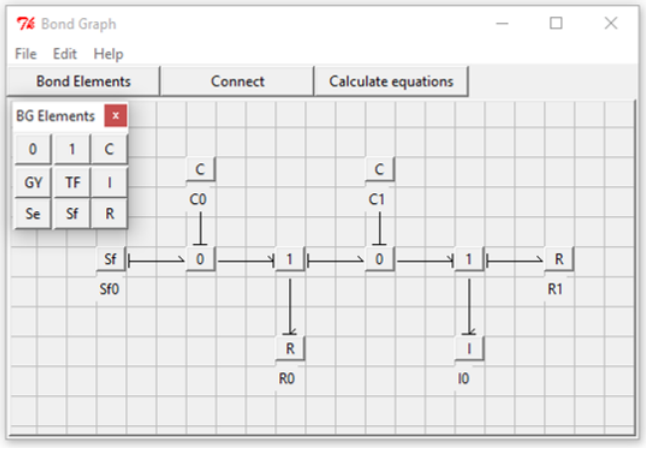

Figure 4.8: Hydraulic System Bond Graph

4.4 Electro-Mechanical System

A simple wind turbine generator with an input wind speed, $v(t)$, is depicted in Fig 4.10.

The bond graph of the system is shown in Fig 4.11 and the state equations is shown in Fig 4.12

```
=============== to i= 1 ================
        d           e(t)
F(t) - ---(q₁(t)) - ----
        dt          R₀
=============== to i= 5 ================
  d           e(t)   p₇(t)
- ---(q₅(t)) + ---- - ----
  dt          R₀     I₀
=============== to i= 7 ================
d          R₁·p₇(t)   2·q₅(t)   q₁(t)
---(p₇(t)) - -------- + ------- + ----
dt           I₀         C₁       C₀
```

Figure 4.9: Hydraulic System State Equations

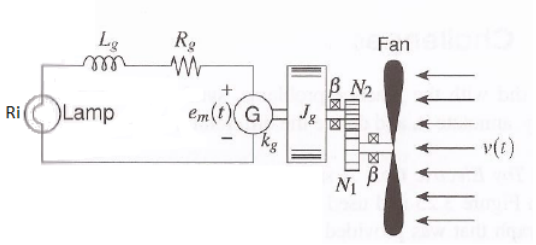

Figure 4.10: Wind Turbine Generator

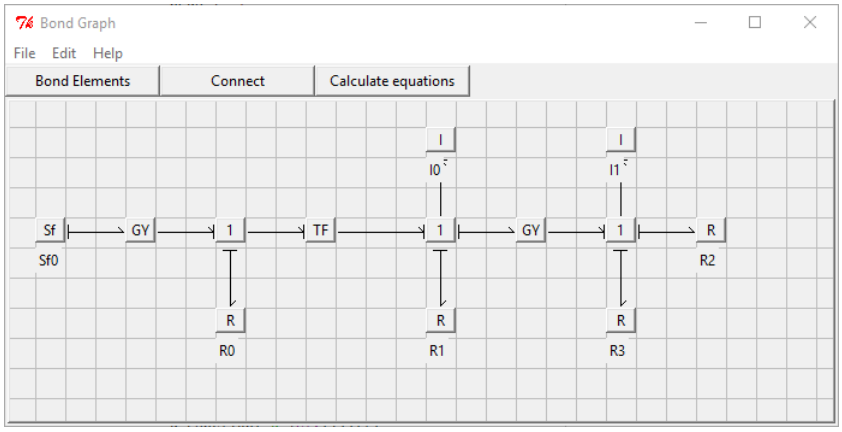

Figure 4.11: Wind Turbine Bond Graph

```
=============== to i= 7 ================

In [90]: runfile('C:/Users/JavierI/Documents/Python
Scripts/ecuaciones.py', wdir='C:/Users/JavierI/Documents/Python
Scripts')
=============== to i= 5 ================
                                      2
          d          r₁   R₁₁·m₀ ·p₅(t)·p₇(t)   R₁₁·p₅(t)·p₇(t)
m₀·r₀·E(t) - ---(p₅(t)) - -- + ---------------- + ---------------
          dt         I₁            I₀                  I₀
=============== to i= 7 ================
          d          R₁₁·p₇(t)   R₃·p₇(t)
r₁·p₅(t) - ---(p₇(t)) + -------- - -------
          dt           I₁          I₁
```

Figure 4.12: Wind Turbine State Equations

Chapter 5 Results, Conclusion and Future Work

Bond Graph is one of the most powerful modeling tools, allowing us to model different systems with different types of energy, namely, mechanical, electrical, hydraulic and the combination of them.

Having a computational tool to draw and get the state equations of a bond graph model is helpful when working with these models, either to teach, study, research or simply apply it as a solution of industrial problems.

The Bond graph tool was developed to provide a graphic interface for drawing and editing of bond graphs, perform automatic causality assignment to the bond graph, and compute the state-space representation of the dynamic system modeled from the bond graph in symbolic form.

The fact that it is an open source software allows anyone to have access to use it and most importantly to help improve its performance through peer contributors.

This software is an excellent starting point for the development of a powerful application.

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 9 00:24:35 2015
"""
.-------------------------------------------------------------------------------.
|                                                                               |
|  Copyright (C) 2005-2014  Cam Farnell                                         |
|                                                                               |
|  This program is free software; you can redistribute it and/or                |
|  modify it under the terms of the GNU General Public License                  |
|  as published by the Free Software Foundation; either version 2               |
|  of the License, or (at your option) any later version.                       |
|                                                                               |
|  This program is distributed in the hope that it will be useful,              |
|  but WITHOUT ANY WARRANTY; without even the implied warranty of               |
|  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the                |
|  GNU General Public License for more details.                                 |
|                                                                               |
|  You should have received a copy of the GNU General Public License            |
|  along with this program; if not, write to the Free Software                  |
|  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA. |
|                                                                               |
`-------------------------------------------------------------------------------'

Modify by: Marihebert Leal
"""


#Set Verbosity to control the display of information messages:
# 2 Displays all messages
# 1 Displays all but dnd_accept and dnd_motion messages
# 0 Displays no messages
Verbosity = 2

#When you drag an existing object on a canvas, we normally make the original
# label into an invisible phantom, and what you are ACTUALLY dragging is
# a clone of the objects label. If you set "LeavePhantomVisible" then you
# will be able to see the phantom which persists until the object is
# dropped. In real life you don't want the user to see the phantom, but
# for demonstrating what is going on it is useful to see it. This topic
# beaten to death in the comment string for Dragged.Press, below.
LeavePhantomVisible = 0

from Tkinter import *
#import Tkinter as tk
import Tkdnd
from numpy import matrix
from sympy import *
from tkMessageBox import *

name="
```

```
sw_line=False #Turns on-off the line plot
sw_BGElem=True #Turns on-off the DnD in the main canvas
#b1 = "up"
line_point='Ini' #Defines the start and the end of the lines
EBold = '' #Stores the Element Bondgraph name
ElemList=[] #Stores elements bondgraph in pairs.
CausList=[] #Stores causalities in pairs.
nEqu=0 #Stores equation number.
EquPrint=[]

def MouseInWidget(Widget,Event):
x = int((Event.x_root - Widget.winfo_rootx())/25)*25
y = int((Event.y_root - Widget.winfo_rooty())/25)*25
return (x,y)
class Dragged:
NextNumber = 0
NextR=0
NextC=0
NextI=0
NextSe=0
NextSf=0
Next1=0
Next0=0
NextTF=0
NextGY=0
def __init__(self):
#When created we are not on any canvas
self.Canvas = None
self.OriginalCanvas = None
#This sets where the mouse cursor will be with respect to our label
self.OffsetX = 20
self.OffsetY = 10
#Assign ourselves a unique number
self.Number = Dragged.NextNumber
self.Junc=0
self.IsConn=False
Dragged.NextNumber += 1
C, R, I, n, t = symbols('C R I n t')
E=Function('E')(t)
F=Function('F')(t)
f=Function('f')(t)
e=Function('e')(t)
q=Function('q')(t)
qd=diff(q,t)
p=Function('p')(t)
pd=diff(p,t)
```

```
#Use the number to build our name
if name == 'C':
self.Name = name + str(Dragged.NextC)
self.Force = q/C #'q/C'
self.Flow = qd #'q_dot'
self.type='C'
Dragged.NextC += 1
elif name == 'R':
self.Name = name + str(Dragged.NextR)
self.Force = R*f #'R*f'
self.Flow = e/R #'e/R'
self.type='R'
Dragged.NextR += 1
elif name == 'I':
self.Name = name + str(Dragged.NextI)
self.Force = pd #'p_dot'
self.Flow = p/I #'p/I'
self.type='I'
Dragged.NextI += 1
elif name == 'Se':
self.Name = name + str(Dragged.NextSe)
self.Force = E #'E(t)'
self.Flow = qd #'q_dot'
self.type='Se'
Dragged.NextSe += 1
elif name == 'Sf':
self.Name = name + str(Dragged.NextSf)
self.Force = pd #'p_dot'
self.Flow = F #'F(t)'
self.type='Sf'
Dragged.NextSf += 1
elif name == '1':
self.Name = name + str(Dragged.Next1)
self.Force = ' '
self.Flow = ' '
self.type='1'
Dragged.Next1 += 1
elif name == '0':
self.Name = name + str(Dragged.Next0)
self.Force = ' '
self.Flow = ' '
self.type='0'
Dragged.Next0 += 1
elif name == 'GY':
self.Name = name + str(Dragged.NextGY)
self.Force = ' '
```

```
self.Flow = ' '
self.type='GY'
Dragged.NextGY += 1
elif name == 'TF':
self.Name = name + str(Dragged.NextTF)
self.Force = ' '
self.Flow = ' '
self.type='TF'
Dragged.NextTF += 1
def dnd_end(self,Target,Event):
#this gets called when we are dropped
if self.Canvas==None and self.OriginalCanvas==None:
#We were created and then dropped in the middle of nowhere, or
# we have been told to self destruct. In either case
# nothing needs to be done and we will evaporate shortly.
return
if self.Canvas==None and self.OriginalCanvas<>None:
self.Canvas = self.OriginalCanvas
self.ID = self.OriginalID
self.Label = self.OriginalLabel
self.Label['text'] = self.OriginalText
self.Label['relief'] = RAISED
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.IDL = self.OriginalIDL
self.Javier = self.OriginalJavier
self.Label['text'] = self.OriginalTextL
self.Javier['relief'] = RAISED
self.Canvas.dnd_enter(self,Event)
return
self.Label.bind('<ButtonPress>',self.Press)
if self.OriginalCanvas:
self.OriginalCanvas.delete(self.OriginalID)
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalCanvas.delete(self.OriginalIDL)
self.OriginalCanvas = None
self.OriginalID = None
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalIDL = None
self.OriginalLabel = None
self.OriginalJavier = None

def Appear(self, Canvas, XY):
if self.Canvas:
#we are already on a canvas; do nothing
return
self.X, self.Y = XY
```

```
Name= self.type
self.Label = Label(Canvas,text=Name,borderwidth=2, relief=RAISED)
#Display the label on a window on the canvas. We need the ID returned by
# the canvas so we can move the label around as the mouse moves.
self.ID = Canvas.create_window(self.X-self.OffsetX, self.Y-self.OffsetY, width = 25,
window=self.Label, anchor="nw")
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.Javier=Label(Canvas, text=self.Name)
self.IDL = Canvas.create_window(self.X-self.OffsetX, self.Y-self.OffsetY+25,
window=self.Javier, anchor="nw")
#Note the canvas on which we drew the label.
self.Canvas = Canvas

def Vanish(self,All=0):
if self.Canvas:
#we have a label on a canvas; delete it
self.Canvas.delete(self.ID)
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.Canvas.delete(self.IDL)
#flag that we are not represented on the canvas
self.Canvas = None
#Since ID and Label are no longer meaningful, get rid of them lest they
#confuse the situation later on. Not necessary, but tidy.
del self.ID
del self.Label
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
del self.IDL
del self.Javier
if All and self.OriginalCanvas:
#Delete label representing us from self.OriginalCanvas
self.OriginalCanvas.delete(self.OriginalID)
self.OriginalCanvas = None
del self.OriginalID
del self.OriginalLabel
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalCanvas.delete(self.OriginalIDL)
del self.OriginalIDL
del self.OriginalJavier

def Move(self,XY):
assert self.Canvas, "Can't move because we are not on a canvas"
self.X, self.Y = XY
self.Canvas.coords(self.ID,self.X-self.OffsetX,self.Y-self.OffsetY)
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.Canvas.coords(self.IDL,self.X-self.OffsetX,self.Y-self.OffsetY+25)
# print "BG type",self.type
```

```
def setJun(self,Val):
self.Junc = Val
# print "**************************self.Junc=",self.Junc,"**************"

def Press(self,Event):
global xold, yold, line_point, xnew, ynew, EBold, ElemList, CausList

# Blab(1, "Dragged.press")
#Save our current label as the Original label
self.OriginalID = self.ID
# print "ID = ",self.ID
self.OriginalLabel = self.Label
self.OriginalText = self.OriginalLabel['text']
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalIDL = self.IDL
self.OriginalJavier = self.Javier
self.OriginalTextL = self.OriginalJavier['text']
self.OriginalCanvas = self.Canvas
#Made the phantom invisible (unless the user asked to see it)
if LeavePhantomVisible:
self.OriginalLabel['text'] = '<phantom>'
self.OriginalLabel['relief']=RAISED
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalJavier['text'] = '<phantom>'
self.OriginalJavier['relief']=RAISED
else:
self.OriginalLabel['text'] = ''
self.OriginalLabel['relief']=FLAT
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.OriginalJavier['text'] = ''
self.OriginalJavier['relief']=FLAT
#Say we have no current label
self.ID = None
self.Canvas = None
self.Label = None
if self.type<>"1" and self.type<>"0" and self.type<>"TF" and self.type<>"GY":
self.IDL = None
self.Javier = None
#Ask Tkdnd to start the drag operation
if Tkdnd.dnd_start(self,Event):
#Save where the mouse pointer was in the label so it stays in the
# same relative position as we drag it around
self.OffsetX, self.OffsetY = MouseInWidget(self.OriginalLabel,Event)
#Draw a label of ourself for the user to drag around
XY = MouseInWidget(self.OriginalCanvas,Event)
self.Appear(self.OriginalCanvas,XY)
```

```
if sw_line:
# print 'line_point',line_point

if line_point=='Ini':
xold,yold = XY
# yold = XY[1]
line_point='End'
EBold = self.Name
# print 'X=',XY[0],'Event.x=',Event.x,'X_old=',xold
else:
# print 'sw_line=',sw_line
xnew,ynew=XY
# ynew=XY[1]
causal=[0,0]
# print 'in b1up xold=',xold
# print 'in b1up xnew=',xnew
if xold<xnew and yold==ynew:
self.OriginalCanvas.create_line(xold+27,yold+12,xnew-2,yold+12,smooth=TRUE) #Draw the
horizontal line (-->)
self.OriginalCanvas.create_line(xnew-2,yold+12,xnew-7,yold+7,smooth=TRUE) #Draw the half
arrow
if EBold[0]=="1" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1: #Case: 1 |->
EBG
self.OriginalCanvas.create_line(xold+27,yold+6,xold+27,yold+18,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1": #When two "1" are connected
self.Junc=1
elif self.Name[0]=="1" and self.Junc==1: #Case: EBG ->| 1
self.OriginalCanvas.create_line(xnew-2,yold+6,xnew-2,yold+18,smooth=TRUE)
causal=[0,1]
elif self.Name[0]=="0" and self.Junc==1: #Case: EBG |-> 0
self.OriginalCanvas.create_line(xold+27,yold+6,xold+27,yold+18,smooth=TRUE)
causal=[0,1]
elif EBold[0]=="0" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1: #Case: 0 ->|
EBG
self.OriginalCanvas.create_line(xnew-2,yold+6,xnew-2,yold+18,smooth=TRUE)
causal=[0,1]
if self.Name[0]=="0": #When two "0" are connected
self.Junc=1
else:
# print "################### START TEST ###################################"
if EBold[0]=="I" or self.Name[0]=="C" or EBold[0]+EBold[1]=="Sf": #Case: EBG |-> EBG
# print "==================== FIRST IF ================================"
self.OriginalCanvas.create_line(xold+27,yold+6,xold+27,yold+18,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1": #Set Junc variable in 1, so the rest of the juction are defined
```

```
self.Junc=1
elif EBold[0]=="0": #Set Junc variable in 1, so the rest of the juction are defined
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
# print "Element=",TargetWidget_TargetObject.ObjectDict[EBold].Junc
elif EBold[0]=="C" or self.Name[0]=="I" or EBold[0]+EBold[1]=="Se": #Case: EBG ->| EBG
# print "==================== SECOND IF
=============================="
self.OriginalCanvas.create_line(xnew-2,yold+6,xnew-2,yold+18,smooth=TRUE)
causal=[0,1]
if EBold[0]=="1": #Set Junc variable in 1, so the rest of the juction are defined
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
# print "Element Para C=",TargetWidget_TargetObject.ObjectDict[EBold].Junc
elif self.Name[0]=="0": #Set Junc variable in 1, so the rest of the juction are defined
self.Junc=1
# print "Element=",self.Junc
elif EBold[0]=="T" or EBold[0]=="G":
# print "Ebold=",EBold
for i in range(len(ElemList)):
if ElemList[i][1]==EBold:
if EBold[0]=="T":
causal=CausList[i]
else:
causal=[CausList[i][1],CausList[i][0]]
break
# print "causal=",causal
if (causal[0]==1 and EBold[0]=="T") or (causal[1]==0 and EBold[0]=="G"): #Hay que cambiar
la pregunta ya que cuando la causalidad es contraria siempre da mal si es G
# print "Entro en el primer lazo"
self.OriginalCanvas.create_line(xold+27,yold+6,xold+27,yold+18,smooth=TRUE)
else:
# print "Entro en el segundo lazo"
self.OriginalCanvas.create_line(xnew-2,yold+6,xnew-2,yold+18,smooth=TRUE)
elif xold>xnew and yold==ynew:
self.OriginalCanvas.create_line(xold-2,yold+12,xnew+27,yold+12,smooth=TRUE) #Draw the
horizontal line (<--)
self.OriginalCanvas.create_line(xnew+27,yold+12,xnew+32,yold+7,smooth=TRUE) #Draw the
half arrow
if EBold[0]=="1" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1: #Case: EBG <-|
1
self.OriginalCanvas.create_line(xold-2,yold+6,xold-2,yold+18,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1": #When two "1" are connected
self.Junc=1
elif self.Name[0]=="1" and self.Junc==1: #Case: 1 |<- EBG
self.OriginalCanvas.create_line(xnew+27,yold+6,xnew+27,yold+18,smooth=TRUE)
causal=[1,0]
```

```
elif EBold[0]=="0" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1: #Case: EBG
|<- 0
self.OriginalCanvas.create_line(xnew+27,yold+6,xnew+27,yold+18,smooth=TRUE)
causal=[1,0]
elif self.Name[0]=="0" and self.Junc==1: #Case: 0 <-| EBG
self.OriginalCanvas.create_line(xold-2,yold+6,xold-2,yold+18,smooth=TRUE)
causal=[0,1]
else:
if EBold[0]=="I" or self.Name[0]=="C" or EBold[0]+EBold[1]=="Sf":
self.OriginalCanvas.create_line(xold-2,yold+6,xold-2,yold+18,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1": #Set Junc variable in 1, so the rest of the juction are defined
self.Junc=1
elif EBold[0]=="0": #Set Junc variable in 0, so the rest of the juction are defined
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif EBold[0]=="C" or self.Name[0]=="I" or EBold[0]+EBold[1]=="Se":
self.OriginalCanvas.create_line(xnew+27,yold+6,xnew+27,yold+18,smooth=TRUE)
causal=[0,1]
if self.Name[0]=="0": #Set Junc variable in 0, so the rest of the juction are defined
self.Junc=1
# print "Element=",self.Junc
elif EBold[0]=="1": #Set Junc variable in 1, so the rest of the juction are defined
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif EBold[0]=="T" or EBold[0]=="G":
for i in range(len(ElemList)):
if ElemList[i][1]==EBold:
if EBold=="T":
causal=CausList[i]
else:
causal=[CausList[i][1],CausList[i][0]]
break
if (causal[0]==0 and EBold[0]=="T") or (causal[1]==0 and EBold[0]=="G"): #Hay que cambiar
la pregunta ya que cuando la causalidad es contraria siempre da mal si es G
self.OriginalCanvas.create_line(xold-2,yold+6,xold-2,yold+18,smooth=TRUE)
else:
self.OriginalCanvas.create_line(xnew+27,yold+6,xnew+27,yold+18,smooth=TRUE)

elif yold>ynew and xold==xnew:
self.OriginalCanvas.create_line(xold+12,yold-2,xold+12,ynew+27,smooth=TRUE) #Draw the
vertical line (|)
self.OriginalCanvas.create_line(xold+12,ynew+27,xold+17,ynew+32,smooth=TRUE) #Draw the
half arrow (^)
if EBold[0]=="1" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1:
self.OriginalCanvas.create_line(xold+6,yold-2,xold+18,yold-2,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1":
```

```
self.Junc=1
elif self.Name[0]=="1" and self.Junc==1:
self.OriginalCanvas.create_line(xold+6,ynew+27,xold+18,ynew+27,smooth=TRUE)
causal=[0,1]
elif EBold[0]=="0" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1:
self.OriginalCanvas.create_line(xold+6,ynew+27,xold+18,ynew+27,smooth=TRUE)
causal=[0,1]
if self.Name=="0":
self.Junc=1
elif self.Name[0]=="0" and self.Junc==1:
self.OriginalCanvas.create_line(xold+6,yold-2,xold+18,yold-2,smooth=TRUE)
causal=[1,0]
else:
if EBold[0]=="I" or self.Name[0]=="C" or EBold[0]+EBold[1]=="Sf":
self.OriginalCanvas.create_line(xold+6,yold-2,xold+18,yold-2,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1":
self.Junc=1
elif EBold[0]=="0":
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif EBold[0]=="C" or self.Name[0]=="I" or EBold[0]+EBold[1]=="Se":
causal=[0,1]
self.OriginalCanvas.create_line(xold+6,ynew+27,xold+18,ynew+27,smooth=TRUE)
if self.Name[0]=="0":
self.Junc=1
elif EBold[0]=="1":
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif EBold[0]=="T" or EBold[0]=="G":
for i in range(len(ElemList)):
if ElemList[i][1]==EBold:
if EBold=="T":
causal=CausList[i]
else:
causal=[CausList[i][1],CausList[i][0]]
break
if (causal[0]==1 and EBold[0]=="T") or (causal[1]==1 and EBold[0]=="G"): #Hay que cambiar
la pregunta ya que cuando la causalidad es contraria siempre da mal si es G
self.OriginalCanvas.create_line(xold+6,ynew+27,xold+18,ynew+27,smooth=TRUE)
else:
self.OriginalCanvas.create_line(xold+6,yold-2,xold+18,yold-2,smooth=TRUE)

elif yold<ynew and xold==xnew:
self.OriginalCanvas.create_line(xold+12,yold+27,xold+12,ynew-2,smooth=TRUE) #Draw the
vertical line (|)
self.OriginalCanvas.create_line(xold+12,ynew-2,xold+17,ynew-7,smooth=TRUE) #Draw the
half arrow (V)
```

```
if EBold[0]=="1" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1:
self.OriginalCanvas.create_line(xold+6,yold+27,xold+18,yold+27,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1":
self.Junc=1
elif self.Name[0]=="1" and self.Junc==1:
self.OriginalCanvas.create_line(xold+6,ynew-2,xold+18,ynew-2,smooth=TRUE)
causal=[0,1]
elif EBold[0]=="0" and TargetWidget_TargetObject.ObjectDict[EBold].Junc==1:
self.OriginalCanvas.create_line(xold+6,ynew-2,xold+18,ynew-2,smooth=TRUE)
causal=[0,1]
elif self.Name[0]=="0" and self.Junc==1:
self.OriginalCanvas.create_line(xold+6,yold+27,xold+18,yold+27,smooth=TRUE)
causal=[1,0]
else:
if EBold[0]=="I" or self.Name[0]=="C" or EBold[0]+EBold[1]=="Sf":
self.OriginalCanvas.create_line(xold+6,yold+27,xold+18,yold+27,smooth=TRUE)
causal=[1,0]
if self.Name[0]=="1":
self.Junc=1
elif EBold[0]=="0":
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif EBold[0]=="C" or self.Name[0]=="I" or EBold[0]+EBold[1]=="Se":
self.OriginalCanvas.create_line(xold+6,ynew-2,xold+18,ynew-2,smooth=TRUE)
causal=[0,1]
if EBold[0]=="1":
TargetWidget_TargetObject.ObjectDict[EBold].Junc=1
elif self.Name[0]=="0":
self.Junc=1
elif EBold[0]=="T" or EBold[0]=="G":
for i in range(len(ElemList)):
if ElemList[i][1]==EBold:
if EBold=="T":
causal=CausList[i]
else:
causal=[CausList[i][1],CausList[i][0]]
break
if (causal[0]==1 and EBold[0]=="T") or (causal[1]==1 and EBold[0]=="G"): #Hay que cambiar
la pregunta ya que cuando la causalidad es contraria siempre da mal si es G
self.OriginalCanvas.create_line(xold+6,ynew-2,xold+18,ynew-2,smooth=TRUE)
else:
self.OriginalCanvas.create_line(xold+6,yold+27,xold+18,yold+27,smooth=TRUE)

line_point='Ini'
TargetWidget_TargetObject.ObjectDict[EBold].IsConn=True
self.IsConn=True
```

```
ElemList.append([EBold, self.Name])
CausList.append(causal)

class CanvasDnd(Canvas):
"""
A canvas to which we have added those methods necessary so it can
act as both a TargetWidget and a TargetObject.
Use (or derive from) this drag-and-drop enabled canvas to create anything
that needs to be able to receive a dragged object.
"""
def __init__(self, Master, cnf={}, **kw):
if cnf:
kw.update(cnf)
Canvas.__init__(self, Master, kw)
#ObjectDict is a dictionary of dragable object which are currently on
# this canvas, either because they have been dropped there or because
# they are in mid-drag and are over this canvas.
self.ObjectDict = {}


def dnd_accept(self,Source,Event):
#Tkdnd is asking us (the TargetWidget) if we want to tell it about a
# TargetObject. Since CanvasDnd is also acting as TargetObject we
# return 'self', saying that we are willing to be the TargetObject.
# Blab(2, "Canvas: dnd_accept")
return self

#----- TargetObject functionality -----

def dnd_enter(self,Source,Event):
#This is called when the mouse pointer goes from outside the
# Target Widget to inside the Target Widget.
# Blab(1, "Receptor: dnd_enter")
#Figure out where the mouse is with respect to this widget
XY = MouseInWidget(self,Event)
#Since the mouse pointer is just now moving over us (the TargetWidget),
# we ask the DraggedObject to represent itself on us.
# "Source" is the DraggedObject.
# "self" is us, the CanvasDnd on which we want the DraggedObject to draw itself.
# "XY" is where (on CanvasDnd) that we want the DraggedObject to draw itself.
Source.Appear(self,XY)
#Add the DraggedObject to the dictionary of objects which are on this
# canvas.
self.ObjectDict[Source.Name] = Source
def dnd_leave(self,Source,Event):
#This is called when the mouse pointer goes from inside the
```

```
# Target Widget to outside the Target Widget.
# Blab(1, "Receptor: dnd_leave")
#Since the mouse pointer is just now leaving us (the TargetWidget), we
# ask the DraggedObject to remove the representation of itself that it
# had previously drawn on us.
Source.Vanish()
#Remove the DraggedObject from the dictionary of objects which are on
# this canvas
del self.ObjectDict[Source.Name]
def dnd_motion(self,Source,Event):
global b1, xold, yold,line_point
#This is called when the mouse pointer moves withing the TargetWidget.
# Blab(2, "Receptor: dnd_motion")
#Figure out where the mouse is with respect to this widget
XY = MouseInWidget(self,Event)
#Ask the DraggedObject to move it's representation of itself to the
# new mouse pointer location.
# if sw_BGElem and self.ObjectDict[Source.Name].IsConn==False:
Source.Move(XY)
# print "Is Connected = ",self.ObjectDict[Source.Name].IsConn
def dnd_commit(self,Source,Event):
#This is called if the DraggedObject is being dropped on us.
#This demo doesn't need to do anything here (the DraggedObject is
# already in self.ObjectDict) but a real application would
# likely want to do stuff here.
# Blab(1, "Receptor: dnd_commit; Object received= %s"%`Source`)
pass

#----- code added for demo purposes -----

def ShowObjectDict(self,Comment):
"""
Print Comment and then print the present content of our ObjectDict.
"""
print Comment
if len(self.ObjectDict) > 0:
for Name,Object in self.ObjectDict.items():
print ' %s %s'%(Name,Object)
else:
print " <empty>"

class TrashBin(CanvasDnd):
"""
A canvas specifically for deleting dragged objects.
"""
def __init__(self,Master,**kw):
```

```
#Set default height/width if user didn't specify.
if not kw.has_key('width'):
kw['width'] =150
if not kw.has_key('height'):
kw['height'] = 25
CanvasDnd.__init__(self, Master, kw)
#Put the text "trash" in the middle of the canvas
X = kw['width'] / 2
Y = kw['height'] /2
self.create_text(X,Y,text='TRASH')
def dnd_commit(self,Source,Event):
"""
Accept an object dropped in the trash.
Note that the dragged object's 'dnd_end' method is called AFTER this
routine has returned. We call the dragged objects "Vanish(All=1)"
routine to get rid of any labels it has on any canvas. Having done
so, it will, at 'dnd_end' time, allow itself to evaporate. If you
DON'T call "Vanish(All=1)" AND there is a phantom label of the dragged
object on an OriginalCanvas then the dragged object will think it
has been erroniously dropped in the middle of nowhere and it will
resurrect itself from the OriginalCanvas label. Since we are trying
to trash it, we don't want this to happen.
"""
# Blab(1, "TrashBin: dnd_commit")
#tell the dropped object to remove ALL labels of itself.
Source.Vanish(All=1)
#were a trash bin; don't keep objects dropped on us.
self.ObjectDict.clear()

class Caus_Window(Frame):
def __init__(self,Parent,msg1,msg2):

self.Causal=Toplevel(Parent) # Crea una ventana hija
# v1.protocol("WM_DELETE_WINDOW", "onexit") # Elimina la opción de salir para evitar el
error
self.Causal.resizable(0,0) # Evita que se le pueda cambiar de tamaño a la ventana
self.Causal.withdraw() # oculta Caus_Window
self.Causal.attributes("-toolwindow", 1)
self.Causal.minsize(width=180, height=90)
self.Causal.maxsize(width=180, height=90)
self.Causal.title("Causality Assign")

wd=7
Ef_Out = Button(self.Causal, text = msg1, width = wd, command = self.Button1)
Ef_Out.place(x=20,y=20)
```

```
Ef_In = Button(self.Causal, text = msg2, width = wd, command = self.Button2)
Ef_In.place(x=90,y=20)
lbl = Label(self.Causal, text="Select an option")
lbl.place(x=30, y=60)

self.Causal.deiconify()

def Button1(self):
self.Out = 1
self.Causal.destroy()
def Button2(self):
self.Out = 2
self.Causal.destroy()

class PaletElemBG(Frame):
def __init__(self):

new=Toplevel(Root) # Crea una ventana hija
# v1.protocol("WM_DELETE_WINDOW", "onexit") # Elimina la opción de salir para evitar el
error
new.resizable(0,0) # Evita que se le pueda cambiar de tamaño a la ventana
new.withdraw() # oculta v1
new.attributes("-toolwindow", 1)
# new =tk.Frame.__init__(self)
# new = Toplevel(self)
new.title("BG Elements")

wd=3
Zeros = Button(new, text = '0', width = wd)
Zeros.grid(row=1,column=1, padx=2, pady=2)
# Zeros.pack()
Zeros.bind('<ButtonPress>',lambda Event, n='0':on_dnd_start(Event,n))

Ones = Button(new, text = '1', width = wd)
Ones.grid(row=1,column=2)
# Ones.pack(side=RIGHT)
Ones.bind('<ButtonPress>',lambda Event, n='1':on_dnd_start(Event,n))

Girator = Button(new, text = 'GY', width = wd)
Girator.grid(row=2,column=1)
Girator.bind('<ButtonPress>',lambda Event, n='GY':on_dnd_start(Event,n))

Trans = Button(new, text = 'TF', width = wd)
Trans.grid(row=2,column=2)
Trans.bind('<ButtonPress>',lambda Event, n='TF':on_dnd_start(Event,n))
```

```
Seffort = Button( new, text = 'Se', width = wd)
Seffort.grid(row=3,column=1)
# Inertia.pack(side=BOTTOM)
Seffort.bind('<ButtonPress>',lambda Event, n='Se':on_dnd_start(Event,n))

Sflow = Button(new, text = 'Sf', width = wd)
Sflow.grid(row=3,column=2)
# Capacitor.pack(side=RIGHT)
Sflow.bind('<ButtonPress>',lambda Event, n='Sf':on_dnd_start(Event,n))

Inertia = Button(new, text = 'I', width = wd)
Inertia.grid(row=2,column=3)
# Inertia.pack(side=BOTTOM)
Inertia.bind('<ButtonPress>',lambda Event, n='I':on_dnd_start(Event,n))

Capacitor = Button(new, text = 'C', width = wd)
Capacitor.grid(row=1,column=3)
# Capacitor.pack(side=RIGHT)
Capacitor.bind('<ButtonPress>',lambda Event, n='C':on_dnd_start(Event,n))

Resistor = Button(new, text='R', width = wd)
Resistor.grid(row=3,column=3)
# Resistor.pack(side=RIGHT)
Resistor.bind('<ButtonPress>',lambda Event, n='R':on_dnd_start(Event,n))

# new.button = tk.Button(new, text = "Close Window", width = 25, command =
self.close_window(new) )
# new.button.pack(side=BOTTOM)
new.deiconify()

def close_window(self,new):
new.withdraw() # oculta v1
# self.destroy()


def b1down(event):
global sw_line, line_point
# sw_line=False
line_point='Ini'


if __name__ == "__main__":

def on_dnd_start(Event,n):
"""
This is invoked by InitiationObject to start the drag and drop process
```

```
"""
global name, sw_line, sw_BGElem

#Create an object to be dragged
# print "clicked at", Event.x, Event.y
name = n
sw_line=False
sw_BGElem=True

ThingToDrag = Dragged()
#Pass the object to be dragged and the event to Tkdnd
Tkdnd.dnd_start(ThingToDrag,Event)

def ShowObjectDicts():
"""
Some demo code to let the user see what ojects we think are
on each of the three canvases.
"""
TargetWidget_TargetObject.ShowObjectDict('UpperCanvas')
TargetWidget_TargetObject2.ShowObjectDict('LowerCanvas')
Trash.ShowObjectDict('Trash bin')
# print '----------'

def Call_Caus():
Caus_Window()
def new_window():
global sw_line, sw_BGElem
PaletElemBG()
sw_line=False
sw_BGElem=True

def CreateLine():
global sw_line, sw_BGElem
sw_line=True
sw_BGElem=False
def CalcEq():
global nEqu, Ft, EqIn, EquPrint

# print "In CalEq:"
# print "Element=",TargetWidget_TargetObject.ObjectDict
# print "Element List=",ElemList
# print "Causality List=",CausList
s=0
for i in range(len(ElemList)): #checks all elements have theirs causality
s += sum(CausList[i])
# print "Sumatoria=",s
```

```
if s<>len(ElemList): #If there are elements without causality
# showerror("Warning", "Sorry, there are some elements without assigned causality")
showwarning("Warning", "Sorry, there are some elements without assigned causality")
AsigCausal()
# d=Caus_Window(Root)
# Root.wait_window(d.Causal)
# print "******************** d=",d.Out,"******************"
for i in range(len(ElemList)): #If all elements have causality
if ElemList[i][0][0]=="S":
BG_Ini=TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Name
BG_junc=ElemList[i][1]
# print "BG_Ini= ",BG_Ini," and BG_junc=",BG_junc
break
# suma=BG_Ini.Force + "="
# sumaE=[]
numElem=len(ElemList)-1 #Number of elemnts without the sources (Se or Sf)
# M,b,r=CalPartial(numElem,BG_Ini,BG_junc)
Ft=[0]*len(ElemList)
nEqu = 0
EqIn=[0]*len(ElemList)
F,Q=CalPartial(numElem,BG_Ini,BG_junc,0)
# print "-.-.-.-.-.-.-. pprint for F -.-.-.-.-.-.-."
# pprint(F)
# print "-.-.-.-.-.-.-. pprint for Q -.-.-.-.-.-.-."
# pprint(Q)
EquPrint=[]
for i in range(len(ElemList)):
if Ft[i]<>0:
print "================ to i=",i," ================"
EquPrint.append(Eq(Ft[i],0))
pprint(Eq(Ft[i],0))
preview(EquPrint)
# print suma
# print 'M=',M
# print 'b=',b
# print 'r=',r
# printMatrix(M,b,r)
def AsigCausal():
for i in range(len(ElemList)):
# print "Par de elementos=",CausList[i]
if sum(CausList[i])<>1:
x1=TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].X
x2=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].X
y1=TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Y
y2=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Y
if ElemList[i][0][0]=="1" or ElemList[i][0][0]=="0":
```

```
Ini_Text=ElemList[i][0][0]
End_Text=ElemList[i][1]
elif ElemList[i][0][0]=="S" or ElemList[i][0][0]=="T" or ElemList[i][0][0]=="G":
Ini_Text=ElemList[i][0][0]+ElemList[i][0][1]
End_Text=ElemList[i][1]
elif ElemList[i][1][0]=="1" or ElemList[i][1][0]=="0":
Ini_Text=ElemList[i][0]
End_Text=ElemList[i][1][0]
elif ElemList[i][1][0]=="S" or ElemList[i][1][0]=="T" or ElemList[i][1][0]=="G":
Ini_Text=ElemList[i][0]
End_Text=ElemList[i][1][0]+ElemList[i][1][1]
else:
Ini_Text=ElemList[i][0]
End_Text=ElemList[i][1]
# if ElemList[i][0][0]=="R" or ElemList[i][0][0]=="I" or ElemList[i][0][0]=="C":
asktext_1=Ini_Text+' -->| '+End_Text
# asktext_1=ElemList[i][0]+'-->|'+ElemList[i][1]
# elif ElemList[i][1][0]=="R" or ElemList[i][1][0]=="I" or ElemList[i][1][0]=="C":
asktext_2=Ini_Text+' |--> '+End_Text
# asktext_2=ElemList[i][0]+'|-->'+ElemList[i][1]
d=Caus_Window(Root, asktext_1,asktext_2)
Root.wait_window(d.Causal)
# if askyesno('Assign Causality', asktext):
if d.Out==1:
# print "============> OK WAS PRESSED"
if x1==x2:
if y1>y2:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x2+6,y2+27,x2+18,y2+27,smooth=TRUE)
CausList[i]=[0,1]
# else:
# TargetWidget_TargetObject.create_line(x1+6,y1-2,x1+18,y1-2,smooth=TRUE)
else:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x2+6,y2-2,x2+18,y2-2,smooth=TRUE)
CausList[i]=[0,1]
# else:
# TargetWidget_TargetObject.create_line(x1+6,y1+27,x1+18,y1+27,smooth=TRUE)
else:
if x1>x2:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x2+27,y2+6,x2+27,y2+18,smooth=TRUE)
CausList[i]=[0,1]
# else:
# TargetWidget_TargetObject.create_line(x1-2,y1+6,x1-2,y1+18,smooth=TRUE)
else:
```

```
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x2-2,y2+6,x2-2,y2+18,smooth=TRUE)
CausList[i]=[0,1]
# else:
# TargetWidget_TargetObject.create_line(x1+27,y1+6,x1+27,y1+18,smooth=TRUE)
else:
# print "============> CANCEL WAS PRESSED"
if x1==x2:
if y1>y2:
# if ElemList[i][1][0]=='R':
# TargetWidget_TargetObject.create_line(x2+6,y2+27,x2+18,y2+27,smooth=TRUE)
TargetWidget_TargetObject.create_line(x1+6,y1-2,x1+18,y1-2,smooth=TRUE)
CausList[i]=[1,0]
# else:
# TargetWidget_TargetObject.create_line(x2+6,y2+27,x2+18,y2+27,smooth=TRUE)
# TargetWidget_TargetObject.create_line(x1+6,y1-2,x1+18,y1-2,smooth=TRUE)
else:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x1+6,y1+27,x1+18,y1+27,smooth=TRUE)
CausList[i]=[1,0]
# TargetWidget_TargetObject.create_line(x2+6,y2-2,x2+18,y2-2,smooth=TRUE)
# else:
# TargetWidget_TargetObject.create_line(x2+6,y2-2,x2+18,y2-2,smooth=TRUE)
# TargetWidget_TargetObject.create_line(x1+6,y1+27,x1+18,y1+27,smooth=TRUE)
else:
if x1>x2:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x1-2,y1+6,x1-2,y1+18,smooth=TRUE)
CausList[i]=[1,0]
# TargetWidget_TargetObject.create_line(x2+27,y2+6,x2+27,y2+18,smooth=TRUE)
# else:
# TargetWidget_TargetObject.create_line(x2+27,y2+6,x2+27,y2+18,smooth=TRUE)
# TargetWidget_TargetObject.create_line(x1-2,y1+6,x1-2,y1+18,smooth=TRUE)
else:
# if ElemList[i][1][0]=='R':
TargetWidget_TargetObject.create_line(x1+27,y1+6,x1+27,y1+18,smooth=TRUE)
CausList[i]=[1,0]
# TargetWidget_TargetObject.create_line(x2-2,y2+6,x2-2,y2+18,smooth=TRUE)
# else:
# TargetWidget_TargetObject.create_line(x2-2,y2+6,x2-2,y2+18,smooth=TRUE)
# TargetWidget_TargetObject.create_line(x1+27,y1+6,x1+27,y1+18,smooth=TRUE)

# print "Elem[0]=",TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].X
# print "Elem[1]=",TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].X

def CalPartial(numElem,BG_Ini,BG_junc,Indice):
```

```
#numElem : Number of Elements
#BG_Ini : Initial element before call the routine
#BG_Junc : Junction Element (0, 1, Transformer or Girator)
#Indice : Inidice number to start the for loop
global nEqu, Ft
# global nEqu, Ft, EqIn

# print " BG_junc=",BG_junc
sumaE1=""
sumaF1=""
if BG_junc[0]=="1":
sumaE=""
# sumaE=BG_Ini.Force #+ "="
sumaF=""
# sumaF=BG.Flow
else:
sumaE=""
# sumaE=BG_Ini.Force #+ "="
sumaF=""
# sumaF=BG_Ini.Flow
# print "=+=+=+=+=+=+=+=+=+=+=+ sumaF=",sumaF,"=+=+=+=+=+=+=+=+=+=+=+"
# print " BG_junc=",BG_junc
# Lineas a borrar ##############
M=[[]]
M[0]=['0']*(numElem+3)
b=[0]*(numElem+3)
if BG_junc=="1":
b[numElem]='f'
else:
b[numElem]='e'
b[numElem+1]='p'
b[numElem+2]='q'
r=[0]*(numElem+3)
r[0]='E'
for i in range(numElem-1+3):
M.append(['0']*(numElem+3))
# sw=0
####################################
I,C,R,t=symbols('I C R t')
f=Function('f')(t)
e=Function('e')(t)
E=Function('E')(t)
F=Function('F')(t)
swGY=False
Ef=[]
Fl=[]
```

```
# EqIn=[0]*len(ElemList)
for i in range(Indice,len(ElemList)):
# print "{{{{{{{{{{{{{{{{ ElemList[i][0][0]=",ElemList[i][0],"
BG_junc",BG_junc,"}}}}}}}}}}}}}}}}"
# if ElemList[i][0][0]!="S" or ElemList[i][0][0]!="1" or ElemList[i][0][0]!="0":
if ElemList[i][0][0]!=BG_Ini:
S10=True
else:
S10=False
if ElemList[i][0][1]=="e":
sumaE=E
# print "NUEVONUEVONUEVONUEVO STATING SUMAE=", sumaE
if ElemList[i][0][1]=="f":
sumaF=F
# print "NUEVONUEVONUEVONUEVO STATING SUMAF=", sumaF
# print "{{{{{{{{{{{{{{{{ S10=",S10,"}}}}}}}}}}}}}}}}"
JAV=0
# if ElemList[i][0]==BG_junc or (ElemList[i][1]==BG_junc and ElemList[i][0][0]!="S"):
if ElemList[i][0]==BG_junc or (ElemList[i][1]==BG_junc and S10):
# print "########################
ElemList[i][0]",ElemList[i][0],"BG_junc=",BG_junc,"S10",S10
if ElemList[i][0]==BG_junc:
# print "++++++++++++++++++++++ SECOND TETS IN FIRST LOOP
======================="
# print "++++++++++++++++++++++ i=",i,"======================="
if BG_junc[0]=="1":
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='C':
# sumaF=sumaF+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
C=Symbol('C'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
q=Function('q'+str(i))(t)
qd=diff(q)
e=q/C
# TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=qd
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=qd
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=q/C
Ef.append(q/C)
Fl.append(qd)
# TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=q/C
# print "ooooooooooooo IS A 1-->C ooooooooooooooooo"
if CausList[i][0]==1:
nEqu += 1
EqIn[i]=1
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='R':
R=Symbol('R'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
if CausList[i][1]==0:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=R*f
```

```
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=""
else:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=""
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=e/R
Ef.append(R*f)
Fl.append(e/R)
# print "ooooooooooooo IS A 1-->R ooooooooooooooooo"
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='I':
I=Symbol('I'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
p=Function('p'+str(i))(t)
pd=diff(p)
f=p/I
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=p/I
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=pd
Fl.append(p/I) #Revisar si esto se puede borrar
Ef.append(pd)
# print "ooooooooooooo IS A 1-->I ooooooooooooooooo"
# indDel=i
if CausList[i][1]==1:
nEqu += 1
EqIn[i]=1
# fi=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
# if sumaF==":
# sumaF=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
# else:
# sumaF=sumaF-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='0' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='1':
Ftp,
Qtp=CalPartial(numElem,TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Name,Target
Widget_TargetObject.ObjectDict[ElemList[i][1]].Name,i)
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='0':
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=Ftp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=""
else:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=""
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=Qtp

if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='TF' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='GY':
# print "BG_Junc=",BG_junc," New
BG_Junc",TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name
Ftp,
Qtp=CalPartial(numElem,TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name,Target
Widget_TargetObject.ObjectDict[ElemList[i+1][1]].Name,i+1)
# print "-------------------------- RETURNED VARIABLES ----------------------------"
```

```
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='TF':
m=Symbol('m'+str(TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[2]))
# print "m=", pprint(m)
# print "Ftp", pprint(Ftp)
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=m*Ftp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=Qtp/m
else:
r=Symbol('r'+str(TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[2]))
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=r*Qtp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=""
swGY = True
# print "Force in 1=",TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# print "SumaE in '1' BEFORE=",sumaE
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force<>"":
if sumaE=="":
if BG_Ini[0]=="G":
sumaE = -TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
else:
sumaE = TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# sumaE1=sumaE
else:
if nEqu>2:
sumaE1 = sumaE
else:
sumaE1 = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="TF" or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="GY":
sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
else:
sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# print "SumaE in '1' AFTER=",sumaE
# print "SumaF in '1' BEFORE=",sumaF
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow<>"":
if sumaF==":
sumaF=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
else:
if nEqu>2:
sumaF1 = sumaF
else:
sumaF1 = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="TF" or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="GY":
sumaF = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
else:
sumaF = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
```

```
# sumaF=sumaF-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
# print "SumaF in '1' AFTER=",sumaF
else:
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='I':
# sumaE=sumaE-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# print "ooooooooooooo IS A 0-->I ooooooooooooooooo"
I=Symbol('I'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
p=Function('p'+str(i))(t)
pd=diff(p)
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=p/I
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=pd
Fl.append(p/I)
Ef.append(pd)
# indDel=i
if CausList[i][1]==1:
nEqu += 1
EqIn[i]=1
# if sumaE=="":
# sumaE=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# else:
# sumaE=sumaE-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='C':
# print "ooooooooooooo IS A 0-->C ooooooooooooooooo"
C=Symbol('C'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
q=Function('q'+str(i))(t)
qd=diff(q)
e=q/C
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=qd
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=q/C
Ef.append(q/C)
Fl.append(qd)
if CausList[i][0]==1:
nEqu += 1
EqIn[i]=1
# if sumaE=="":
# sumaE=TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# else:
# sumaE=sumaE-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='R':
R=Symbol('R'+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[1])
if CausList[i][1]==0:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=R*f
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=""
else:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=""
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=e/R
```

```
# TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=""
# TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=e/R
# print "ooooooooooooo IS A 0-->R oooooooooooooooooo"
# print "Force in 0=",TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='0' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='1':
Ftp,
Qtp=CalPartial(numElem,TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name,Target
Widget_TargetObject.ObjectDict[ElemList[i][1]].Name,i)
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='0':
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=Ftp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=-""
else:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=""
if Qtp<>"":
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=Qtp
else:
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=""
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='TF' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='GY':
# print "BG_Junc=",BG_junc," New
BG_Junc",TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name
Ftp,
Qtp=CalPartial(numElem,TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name,Target
Widget_TargetObject.ObjectDict[ElemList[i+1][1]].Name,i+1)
# print "-------------------------- RETURNED VARIABLES ----------------------------"
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='TF':
m=Symbol('m'+str(TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[2]))
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=m*Ftp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=Qtp/m
else:
r=Symbol('r'+str(TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Name[2]))
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force=r*Qtp
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow=Ftp/r
swGY = True
# print "SumaE in '0' BEFORE=",sumaE
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force<>"":
if sumaE==":
sumaE = TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
else:
if nEqu>2:
sumaE1 = sumaE
else:
sumaE1 = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="TF" or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="GY":
```

```
sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
else:
sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="TF" or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="GY":
# sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# else:
# sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
# sumaE = sumaE - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force

# print "SumaE in '0' AFTER=",sumaE
# print "SumaF in '0' BEFORE=",sumaF
if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow<>"":
if sumaF==":
sumaF = -TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
else:
if nEqu>2:
sumaF1 = sumaF
else:
sumaF1 = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow

if TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="TF" or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=="GY":
sumaF = sumaF + TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
else:
sumaF = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
# sumaF = sumaF - TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
# print "SumaF in '0' AFTER=",sumaF
# M[0][i-1]="1"
elif JAV==1:
# print "++++++++++++++++++++++ SECOND TETS IN SECOND LOOP
======================="
if BG_junc[0]=="1":
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='C':
# sumaF=sumaF+TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Flow
TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow=qd
if CausList[i][1]==1:
nEqu += 1
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='R':
R=Symbol('R'+str(i))
TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Force=R*f
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='I':
p=Function('p'+str(i))(t)
pd=diff(p)
TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow=p/I
# fi=TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow
```

```
if sumaF==":
sumaF=TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow
else:
sumaF=sumaF+TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow
TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Force=pd
if CausList[i][0]==1:
nEqu += 1
sumaE = sumaE + TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Force

else:
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type<>'1':
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='I':
# sumaE=sumaE-TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].Force
if CausList[i][0]==1:
nEqu += 1
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='C':
q=Function('q'+str(i))(t)
qd=diff(q)
TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow=qd
sumaE=sumaE+TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Force
if CausList[i][1]==1:
nEqu += 1
# print "<<<<<<<<<<<< SUMAF=",sumaF,">>>>>>>>>>>>>>>"
# print "
Flow=",TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow,">>>>>>>>>>>>>>>"
# print " ElemList[i][0]=",ElemList[i][0],">>>>>>>>>>>>>>>"
if sumaF==":
sumaF = TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow
else:
sumaF = sumaF + TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Flow

# print "Number of equation=",nEqu
swForce=True
swFlow=True
for i in range(Indice,len(ElemList)):
# print "**************************** STARTS TO SAVE DATA IN Ft Variable:
**********************************"
# print "i=",i," BG_Ini=",BG_Ini," BG_Junc=",BG_junc," EqIn=",EqIn[i]
if EqIn[i]==1 and BG_Ini[0]<>"T":
if Ft[i]==0:
# print "Ft is cero, I'm going to assign values"
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='I' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='I':
# Ft.append(sumaE)
if BG_Ini[0]<>"G":
if swGY:
```

```
Ft[i]=sumaF
else:
Ft[i]=sumaE
swForce=False
else:
# print "i=",i,"Ft=",Ft[1],"sumaE=",sumaE,"sumaF=",sumaF
if swForce==False and swGY:
Ft[i]=sumaF
else:
Ft[i]=sumaE
swForce=False
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='C' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='C':
# Ft.append(sumaF)
if BG_Ini[0]<>"G":
Ft[i]=sumaF
swFlow=False
else:
Ft[i]=sumaE
swFlow=False
# print "Ft[",i," ]=",Ft[i]
else:
# print "Ft is not cero, I'm going to add values"
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='I' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='I':
# print "swForce=",swForce
if BG_Ini[0]<>"G":
Ft[i]=Ft[i]+sumaE
swForce=False
else:
if swForce:
# Ft.append(sumaE)
Ft[i]=Ft[i]+r*sumaF
swForce=False
if TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].type=='C' or
TargetWidget_TargetObject.ObjectDict[ElemList[i][1]].type=='C':
# print "swFlow",swFlow
if BG_Ini[0]<>"G":
if sumaF<>"":
Ft[i]=Ft[i]+sumaF
swFlow=False
else:
if swFlow:
if sumaE<>"":
# Ft.append(sumaF)
Ft[i]=Ft[i]+sumaF
```

```
swFlow=False
# print "Ft[",i," ]=",Ft[i]
EqIn[i]=0
if sumaE1=="":
sumaE1=sumaE
if sumaF1=="":
sumaF1=sumaF
if nEqu>2:
return sumaE1, sumaF1
else:
return sumaE, sumaF

# if nEqu>2:
# sumaE=sumaE+TargetWidget_TargetObject.ObjectDict[ElemList[indDel][1]].Force
# return sumaE, sumaF
def donothing():
filewin = Toplevel(Root)
button = Button(filewin, text="Testing the menu")
button.pack()

def SaveFile():
print TargetWidget_TargetObject.ObjectDict
# with open("Test.bgf", 'w') as Output:
# for i in range(len(ElemList)):
# Output.write(TargetWidget_TargetObject.ObjectDict[ElemList[i][0]].Force)
def MenuCreator():
menubar = Menu(Root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=SaveFile)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=Root.quit)
menubar.add_cascade(label="File", menu=filemenu)
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
editmenu.add_separator()
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)
menubar.add_cascade(label="Edit", menu=editmenu)
helpmenu = Menu(menubar, tearoff=0)
```

```
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)
Root.config(menu=menubar)
def GridCanvas():
x1=0
x2=2000
for k in range(0,2*x2,25):
y1=k
y2=k
TargetWidget_TargetObject.create_line(x1, y1, x2, y2, fill='gray')
y1=0
y2=2000
for k in range(0,y2,25):
x1=k
x2=k
TargetWidget_TargetObject.create_line(x1, y1, x2, y2, fill='gray')

Root = Tk()
Root.title('Bond Graph')
img1 = PhotoImage(file="Happy_face.gif")

#Create a button to act as the InitiationObject and bind it to <ButtonPress> so
# we start drag and drop when the user clicks on it.
#The only reason we display the content of the trash bin is to show that it
# has no objects, even after some have been dropped on it.
TargetWidget_TargetObject = CanvasDnd(Root,relief=RAISED,bd=2)
TargetWidget_TargetObject.pack(side=BOTTOM, expand=YES, fill=BOTH)
button1 = Button( Root, text = "Bond Elements", width = 17, command = new_window )
button1.pack(side=LEFT)

button2 = Button( Root, text = "Connect", width = 17, command = CreateLine )
button2.pack(side=LEFT)

button3 = Button( Root, text = "Calculate equations", width = 17, command = CalcEq )
button3.pack(side=LEFT)


GridCanvas() #Draw grid in canvas
MenuCreator() #Invokes the routine to create the main menu

Root.mainloop()
```

Bibliography

[1] Paynter, H.M. *Analysis and Design of Engineering Systems.* M.I.T. Press, Cambridge, Mass., 1960.

[2] Karnopp, D.C., Margolis,D.L., and Rosenberg, R C. *System Dynamics:Modeling, Simulation, and Control of Mechatronic Systems.* Fifth Edition, John Wiley Sons, New Jersey, 2012.

[3] Kypuros, J.A. *System Dynamics and Control with Bond Graph Modeling.* CRC Press Taylor Francis Group, Boca Raton, FL, 2013.

[4] Mukherjee, A.,Karmakar, R.,Samantaray, A.K.
Bond Graph in Modeling, Simulation and Fault Identification I.K. International Publishing House Pvt. Ltd, 2006.

[5] Rosenberg, R.C.
State-Space Formulation for Bond Graph Models of Multiport System Journal of Dynamic System, Measurement, and Control, Trans.ASME, March 1971,pp 35-40.

[6] A site dedicated to modeling and Simulation of mechatronics Systems using the Bond graph Technology.
`http://www.bondgraph.com`

[7] Modelica and the Modelica Association.

http://www.modelica.org

[8] 20 SIM The power in modeling.

http://www.20sim.com

[9] Model Transforation Tools -MTT .

http://www.sourceforge.net

[10] Open Modelica .

http://www.openmodelica.org

[11] Rapyd-Tk

http://www.bitflipper.ca/rapyd/

[12] GitHub-Rapyd-Tk

https://github.com/camfarnell/rapyd-tk