Visualising the Effects of Non-linearity by Creating Dynamic Causal Diagrams

H. Willem Geert Phaff, Jill H. Slinger

March 26, 2007

Abstract

Even though non-linearity is said to drive the behaviour of system dynamics models, modellers do not have access to techniques that show this happening. The tools at their disposal to present model structure are powerful, but, by nature, static. Formal model analysis methods has made significant progress in explaining how structure drives behaviour, but the connection to standard model conceptualisation techniques have generally not yet been made. This paper presents work that links the results of recent formal model analysis techniques to traditional conceptual diagramming techniques. A prototype visualisation tool is used to create dynamic causal diagrams that display the changing influences of the elements of model structure over a simulation run; it shows the waxing and waning of the influence of different sets of loops.

1 Introduction

One of the strengths of System Dynamics models is their foundation in nonlinearity. This nonlinearity enables the 'shifts in loop dominance' seen in System Dynamics models and drives their behaviour (Forrester, 1987; Richardson, 1984). It does, however, come at a price, as the behaviour of models becomes difficult to explain in all but the simplest examples. Formal methods that analyse the relation between structure and behaviour are still the subject of study and development, and are seen as an important topic of research (Sterman, 2000; Richardson, 1996).

The System Dynamics approachs also brings with it an impressive array of conceptual techniques to represent model structure. For instance, causal diagrams, stock-flow diagrams and influence diagrams (Wolstenholme and Coyle, 1983) serve as tools for communication in traditional model development and group model building.

Recent developments in formal model analysis make a link between tradional causal diagrams and the results of formal model analysis techniques feasible. This is, however, not trivial, as most of the methods in formal model analysis are difficult to apply and interpret. The exception to this seems to be Digest (Mojtahedzadeh et al., 2004), which isolates and displays the dominant loops over different time intervals in a simulation run. This has not been done yet for methods based on eigenvalue elasticity. This paper will focus on these methods. Consequently, we will focus on eigenvalue elasticity analysis.

Attempts have been made at automating eigenvalue elasticity analyses, but, while some of these are succesful (Kampmann and Oliva, 2006), not all of them are generally applicable (Güneralp, 2006). Generalised and automated versions of the methods would be stimulus for the application of these methods. Also, a large part of the infrastructure used for these methods is the same. For instance, most methods make use of model linearisations, while selecting specific points in time at which to analyse the model. A common framework would speed up the automation of individual methods, faciliting development and application.

In addition to this, in spite of its power as an analytical tool (Kampmann and Oliva, 2006; Forrester, 1982; Kampmann, 1996), eigenvalue elasticity analysis in particular is challenging to translate into a system story understandable to the client. An intermediate step in generating these stories is the visualisation of the results of the analysis, which usually is in the form of timeplots of the relative influence of the elements of structure investigated (loops, edges, parameters). A method for displaying the relative influences in language close to the usual conceptual diagrams of System Dynamics models would help the analyst in presenting and understanding the results of the formal analysis.

In this paper, several tools are presented to aid the modeller in formal model analysis, and to link the results thereof back to causal diagrams. It roughly consists of three parts. The first part describes the basis of the framework used to run models and perform analyses. The second part shows utilities to present the structure of the model. The third part links the two together to visualize the results of the analyses in a form related to causal diagrams. Finally, results are discussed and suggestions for further research made.

2 Method

2.1 General Design of the AMBA framework

This section describes the framework designed to perform formal model analysis. The intention of the framework is to be able to consistently perform different model analyses. The framework separates analysis, model and numerical solver, while rigidly defining which information travels between the different parts.

SD software packages currently do not support formal model analysis and it is not easy to automatically obtain the information about the model necessary to perform the analysis[1]. Doing the analyses by hand is no longer feasible for even medium sized models.

The approach presented here has as a specific aim to be broadly applicable and is flexible enough to easily adjust for the different forms of analysis. For instance, it should be possible to replace the Kamp-mann (1996) algorithms with the Güneralp (2006) way of relating elasticities to specific variables. Or, to adjust the granularity of the analysis, or to focus on parameters instead of edges or loops.

2.2 Components

The main requirements for the framework are consistency and flexibility. Consistency in the way of working with different models and analyses; one general approach to multiple forms of analysis. Flexibility in being able to replace parts of the framework and change properties of one part without affecting the other. Lastly, the individual parts are set up as generically as possible, making them applicable to as broad a range of models as possible.

As stated above, the idea central to the framework is to separate the solver, the model and the analysis as much as possible. They are seen as completely independent entities exchanging information (Figure 1), each with their own responsibilities. This enables us to replace and change one of the three without affecting the others. Consequently:

- The integration method and its settings are independent from the model and analysis.
- The model interfaces with the tools for analysis and the integrator so that both can obtain and feed back the information required to run or analyse the model.
- The analysis functions obtain the information necessary for performing their calculations and work as generically as possible.

The structural analysis functions (Oliva, 2004) use a representation of the structure of the model; an adjacency matrix. The output of these function also serves as input for loop based dominance analysis. The dominance analysis functions use linearisations of the model in order to be able to calculate, for instance, eigenvalues and related elasticities. Furthermore, they need references to model parameters for the parameter based variants of analysis.

[1] For instance, Powersim is not able to provide linearisations of a model.

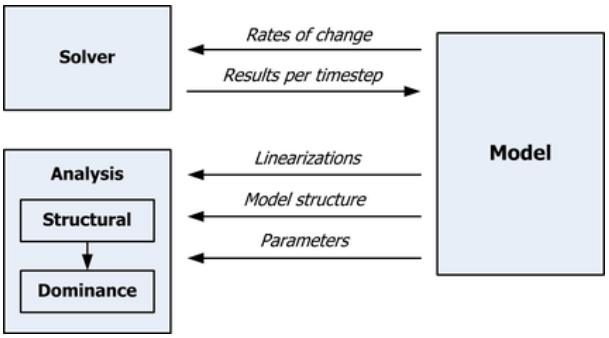


Figure 1: Setup for the framework : Information exchanged between the different parts

2.3 The Model Representation

The purpose of the model representation is to provide the rest of the framework with the necessary information. The model is an outer shell, hiding its internal workings from the rest of the framework. The model provides the data required to run and analyse the model. This includes linearisations and adjacency matrices and net rates of change of the states. Behind it lies the model structure, which consists of a set of linked variables.

The internal structure the model does not matter to the rest of the framework, but we need some form of structure that is able to generate the information required. In essence, the loop based variants of model analysis use a graph perspective on System Dynamics models (Oliva, 2004; Kamp-mann, 1996). However, in general, the graph perspective is incorporated into the analysis only at the very last point. The internal structure presented here uses that perspective from the ground up, ensuring that a System Dynamics model can be perceived as a graph when necessary.

The essence of the presented concept is the hierarchy of the elements in a System Dynamics model. All elements can be seen as vertices in a graph, where the system is the graph. The model elements are divided into variables and and parameters (Figure 2).

A variable is a model element that is dependent on the value of other parts of the model or on time. Each variable should be able to calculate the gains[2] from its predecessors to itself. There are two special forms of variables, namely the state (a.k.a stock or level) and the rate (a.k.a. flow).

This way of representing the model can provides a direct translation to the graph perspective since every element is already defined as a vertex. Secondly, edge gains can be determined by having the destination variable estimate the gains from each of the predecessor to itself. This allows us to linearise the model automatically. Thirdly, rates of change can be determined for each timestep using a recursive method call starting at the states of the model. And, finally, the separate definition of parameters allows us to request these for purposes of analysis. To summarise, this can provide us with the information the rest of the framework needs in order to undertake model analysis.

The current implementation is in Java and is linked to the JUNG library. This implementation is also being applied to hybrid models, interfacing with either DSOL (Jacobs, 2005) or Repast (North

[2]The edge gain is determined by taking the partial derivative of the dependent variable to the independent variable and shows how strongly the dependent variable responds to a perturbation of the independent variable.

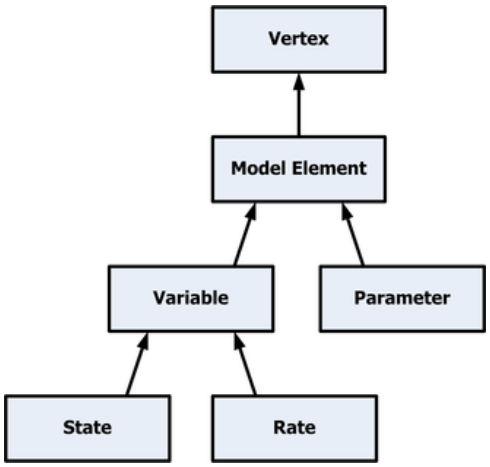


Figure 2: Conceptual Hierarchy of the elements in a System Dynamics model. The arrows denote an inheritance relation, the upper class is more general than the lower.

et al., 2006). The representation interfaces effortlessly with MatLab, opening up the possibility of using MatLab as a scripting tool for analysis such as sensitivity analysis and active nonlinear testing (Appendix A). In addition, a Perl script has been written which enables automatic translation from Vensim files to this implementation.

2.4 The Overall Procedure

The main procedure executes model runs and analyses; linking the separate parts of the framework together. The procedure runs the model until the moment for which we have defined the first analysis. It stops running at that point, performs the analysis, stores the results and continues to run until the next time for analysis. This continues until either the last time for analysis or the last timestep has been reached. During intervals of rapidly changing loop gains, the times for analysis can be closer together. So, both the integration timesteps and the times at which the analysis is performed are completely up to the analyst. For an overview of the procedure see Figure 3.

2.5 The Analysis Functions

These are the functions that implement the formal model analysis by Forrester (1982), Kamp-mann (1996) and Güneralp (2006) and the structural analysis by Oliva (2004)[3]. They have been generalised as far as possible. For instance, the algorithm used by Güneralp and Gertner (2006) to determine an eigenvalue's contribution to the behaviour of a state variable has been translated to a generic method applicable to an nth-order model. This fully automates the approach, although scalability issues still are present. In the current setup, the necessary data can be obtained from the model and the functions perform their calculations.

In line with the setup for the framework, only the required information and the results of the functions are defined.

Table 1: Analysis functions currently present in the framework

Analysis	**Required Input**	**Results**
Loop Eigenvalue Elasticity Analysis	Model Linearisations, Directed Cycle Matrix	Loop elasticities in the form requested
Parameter Eigenvalue Elasticity Analysis	Model Linearisations, Model Parameters	Parameter elasticities in the form requested
Elasticity analysis related to a specific variable	Model Linearisations, Results of the elasticity analysis	Overall elasticity of specific variable to a given parameter or loop

[3]The algorithms from R. Oliva's research page have been used

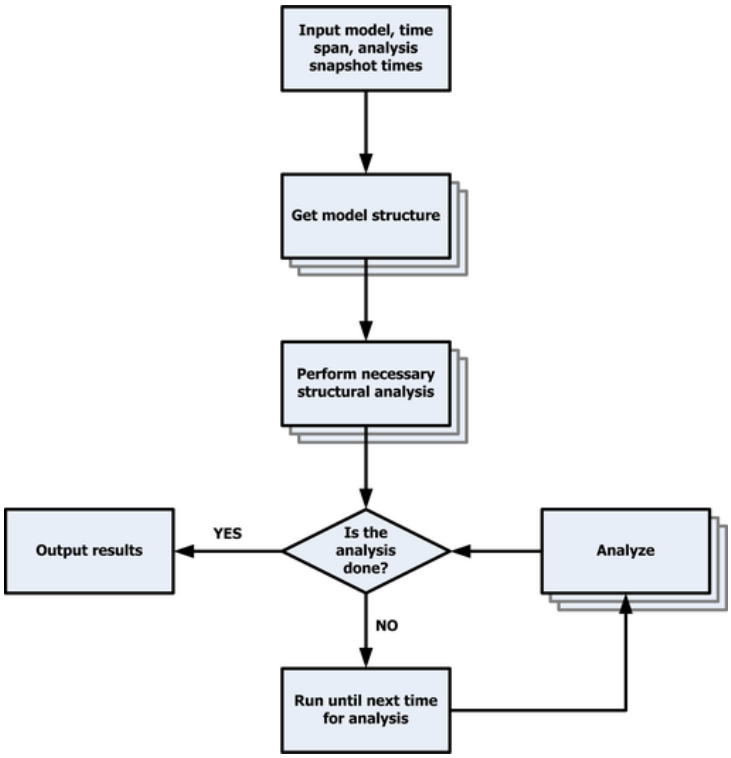


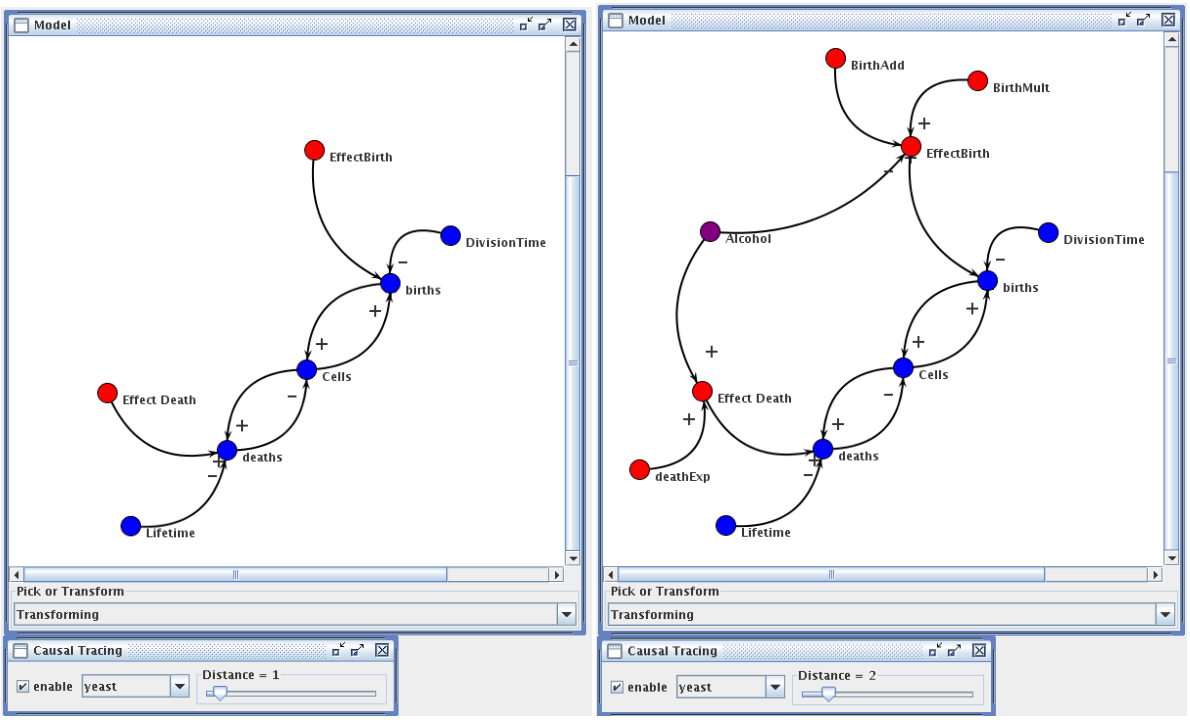
Figure 3: Overview of the main procedure of model analysis

3 Visualizing Structure

Given the complexity and interwovenness of System Dynamics models, the analyst nearly needs to modify the presentation of structure to visualise the model. This paragraph will present ways of visualising only structure, before dynamic causal diagrams are presented. A completed model is visualised, this is not part of the development process.

The visualisation uses the edge gains in the running model to display it as a causal diagram. An edge is a direct link between two model variables, the soure is the independent variable, the destination the dependent variable. The edge gain is determined by taking the partial derivative of the dependent variable to the independent variable and shows how strongly the dependent variable responds to a perturbation of the independent variable. The sign of the causal link is given by the sign of the edge gain, relating this to link polarity (Richardson, 1995). The gain of an inflow is defined as $+1$, an outflow -1. Given the link to formal model analysis the visualisation focuses on the loops present in the model. The tools can manipulate the presentation of loops. As a side-effect, manipulation of submodels and causal tracing is feasible as well. We will show examples of causal tracing and displaying loops.

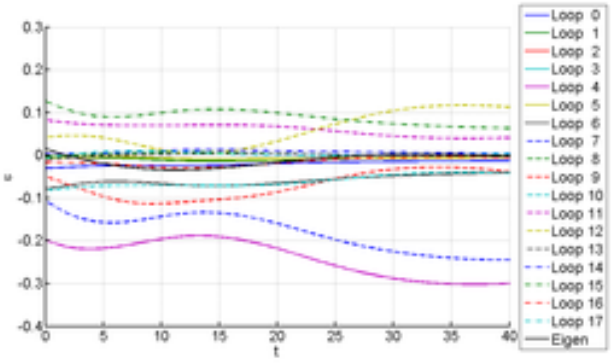
- **Causal Tracing:** With regards to causally tracing the variables, the tool allows for showing the variables at a number of steps away from the object of interest. That is, the analyst can show, for instance, the elements at two steps away from a selected loop, submodel or variable (Figure 4).
- **Displaying Loops:** Most methods of formal model analysis focus on the loops, visualizing these will help in interpreting results. The visualisation accepts a Directed Cycle Matrix (DCM) (Kampmann, 1996) of a set of loops. These loops can then be highlighted, or selected to be the only ones displayed. This facilitates reading the usual plots that are the result of a loop based eigenvalue elasticity analysis. For illustrative purposes, we show a small part of the results of a loop eigenvalue elasticity analysis of an unmodified version of the Market Growth model (Forrester, 1968; Morecroft, 1983) (Figure 5). The model is selected based on its size and how well known it is. The Market Growth model is a classic 10^th order model of an electronics manufacturer. The model shows the consequences of bounded rationality in the interplay of sales growth and production capacitity in a recently opened product market.



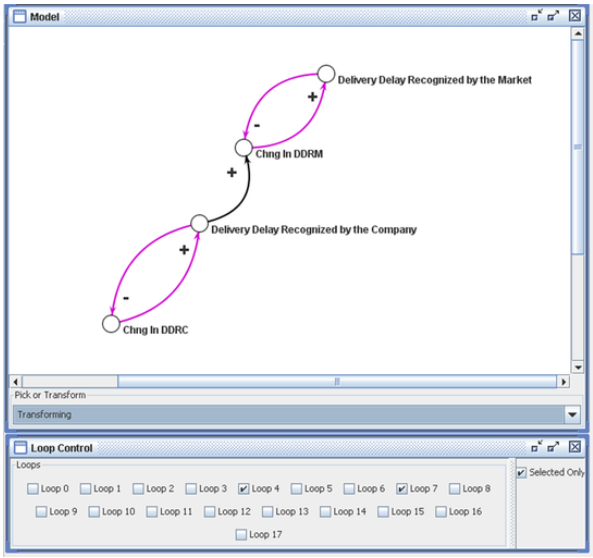
(a) Elements at distance 1

(b) Elements at distance 2

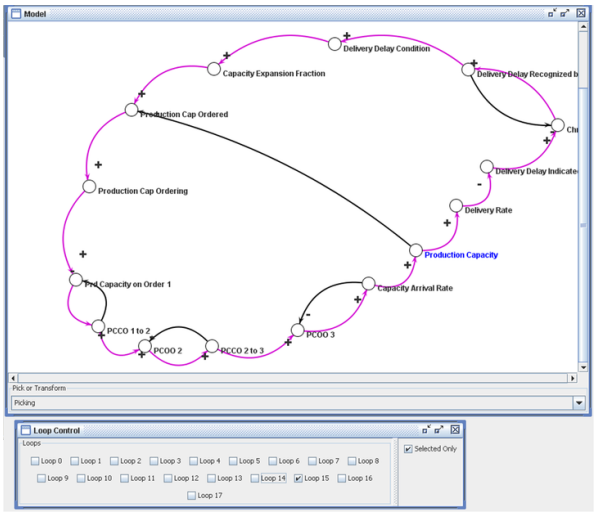
Figure 4: Causal tracing in the Yeast model. Image (a) shows the the variables related to Cells and their births and deaths in blue, with elements at one link away in different colors. Image (b) shows the same submodel but now with elements at two or less edges away.



(a) Elasticity Plot



(b) Loops with large negative elasticitity



(c) Loop with large positive elasticity

Figure 5: Eigenvalue elasticities for a short part of the run of the Market Growth model (a). On the selected eigenvalue, the loops associated with the delay in the market and the company recognizing delivery delay (b) have the strongest dampening influence. The loop with the largest positive elasticity is the unnamed high-order Loop 17.

4 Dynamic Causal Diagrams

As stated, one of the difficulties in using Loop Eigenvalue Elasticity like methods is the interpretation of the results (Phaff, 2006; Kampmann and Oliva, 2006). Of the strategies suggested one of the most prominent ones is to relate the way an edge is drawn in the model to the measure of its influence. Kampmann and Oliva (2006), for instance, suggest controlling the glow of an edge to make it stand out more when it has more influence.

As time progresses the influences and gains of edges change in any non-linear model. The time at which the visualisation is displayed has to be changeable in the visualisation. It is this evolution through time of the relative influence of a particular element of structure that aids in showing the system story.

The danger in this approach, however, is that a complex method is oversimplified into an attractive visualisation. Any tool that uses this must still make explicit that, for instance, when applying loop eigenvalue elasticity, the analyst has to chose an eigenvalue and a particular type of elasticity. At every moment in time, one diagram can be drawn for each eigenvalue. The tool presented here offers the full choice of eigenvalues, types of elasticities and draws the model according to what the analyst selects.

The visualisation is presented using the Yeast model (Saleh, 2002; Güneralp, 2005; Phaff, 2006). The opacity of the links is determined by the relative size of the measure of influence. The color is dependent on the sign of the influence, if the influence is less than zero, the edge is colored blue, if larger than zero, the edge is colored red. We will display the visualisation for the edge gains in the model, the real edge eigenvalue elasticities and the overall edge eigenvalue elasticity. For the visualisation of gains, the opacity for edge i, e_i is calculated as

$$\gamma(e_i) = 0.1 + 0.9\frac{|g(e_i)|}{\max\{|g(e)| \mid e \in E\}} \tag{1}$$

where E is the set of edges in the model, $g(e)$, the gain of an edge. The minimum value for opacity is included to prevent edges with low gains from disappearing completely from the analyst's view. With regard to eigenvalue elasticity (imaginary or real), a quantitative measure of the influence an element of structure has on behaviour, of e_i, to a particular eigenvalue λ_k, $\epsilon_{k,i}$, the opacity is calculated as

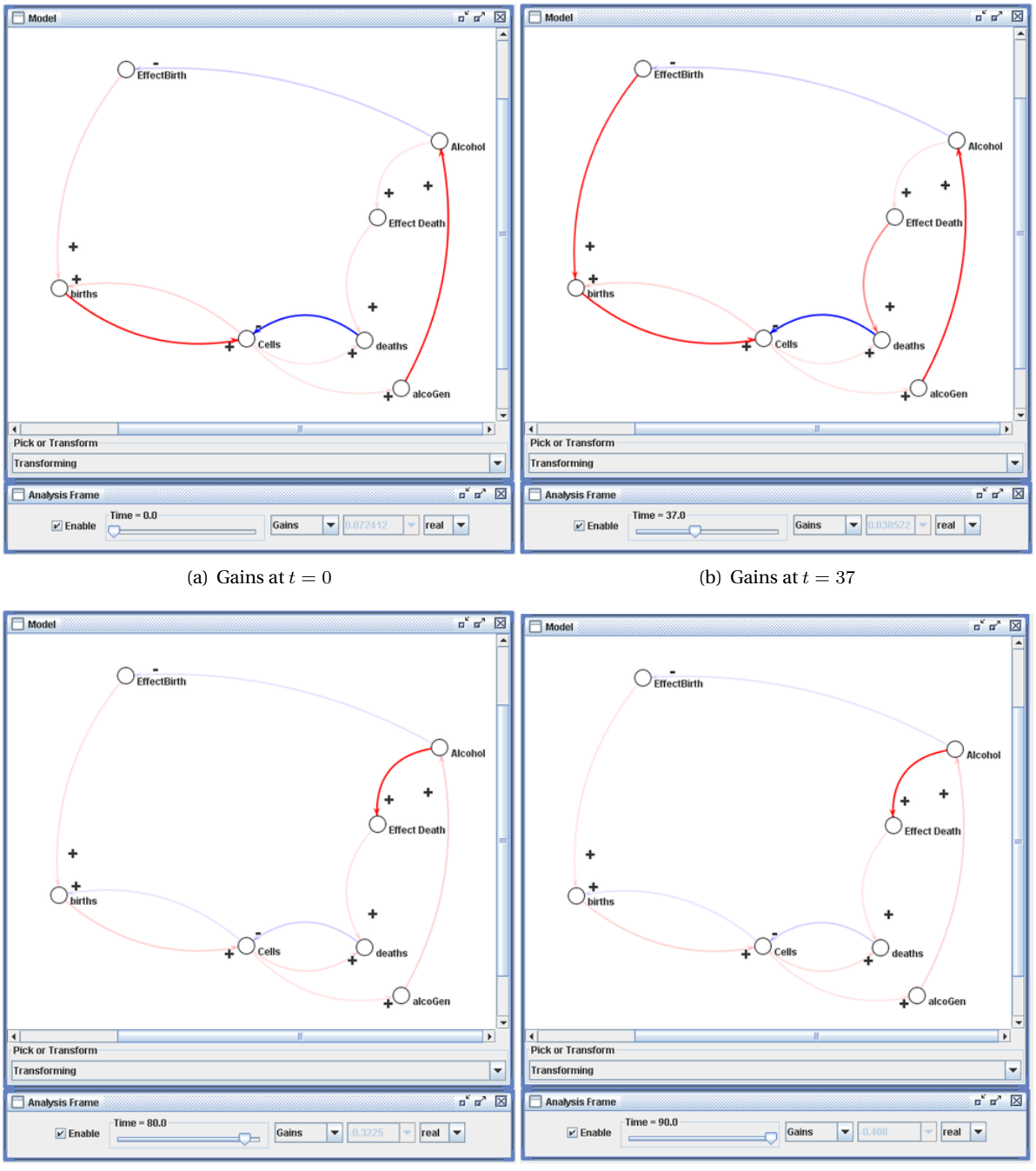
$$\gamma(e_i) = 0.1 + 0.9\frac{|\epsilon_k(e_i)|}{\max\{|\epsilon_k(e)| \mid e \in E\}} \tag{2}$$

In effect, the visualisation shows a causal diagram, where edges (links) change in visibility as their influence waxes and wanes, creating a dynamic causal diagram. This immediately shows how the non-linearities in the model cause the shifts in driving structure. It maps the result of a complex, mathematical method of analysis back to the visual language of the modeller.

The figures below (Figure 6, Figure 7, Figure 8) show how the loops driving the behaviour of the Yeast model change over time. Figure 7 shows the initial relative strength of the loop governing the growth of Cells (Figure 4), followed by the increasing influence of the loops constraining that growth as time progresses and the alcohol concentration rises. At the end of the model run, behaviour is mostly governed by the first order loop responsible for the exponential decline of Cells. In Figure

7 displays the direction of influence of the edges on the real part of the eigenvalues, while Figure 8 shows the absolute measure of influence without taking the sign of the influence into account. As time progresses the visualisation displays the changes in driving structure.

It is these changes in driving structure that show the effects of nonlinearity in System Dynamics models. In linear models the roles of the edges and loops in the system would be constant. As Forrester (1987) and Richardson (1984) said, it is these changes in loop dominance that distinguish nonlinear models from linear models.



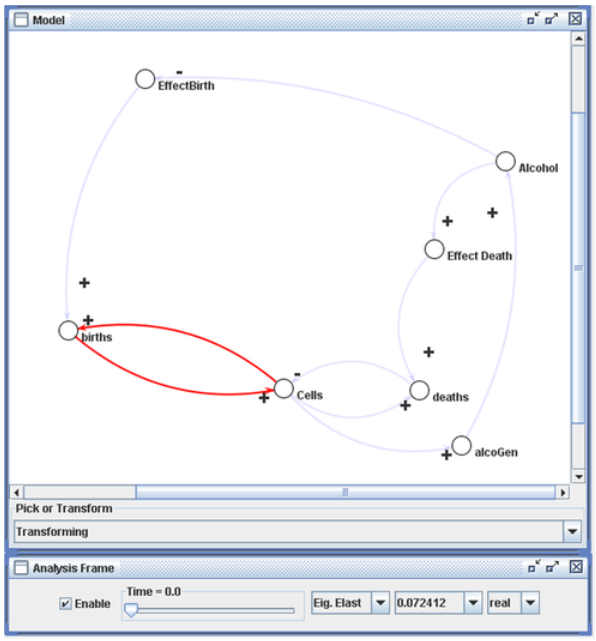
(a) Gains at $t = 0$

(b) Gains at $t = 37$

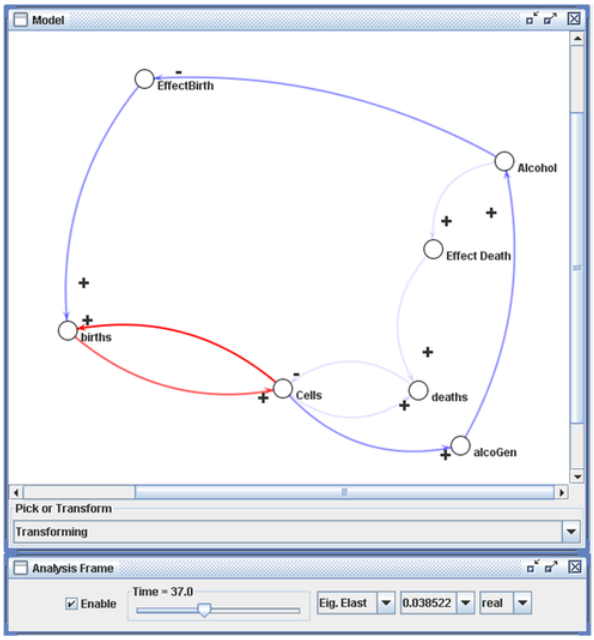
(c) Gains at $t = 80$

(d) Gains at $t = 90$

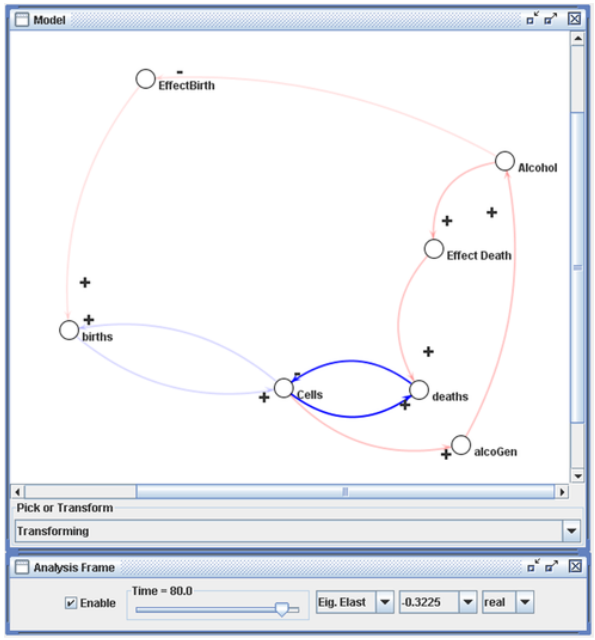
Figure 6: Visualisation of the changing individual edge gains during a simulation. This gives an indication of when a loop is active and the relative strength of the edges therein. Note: in a linear model this visualisation would be constant.



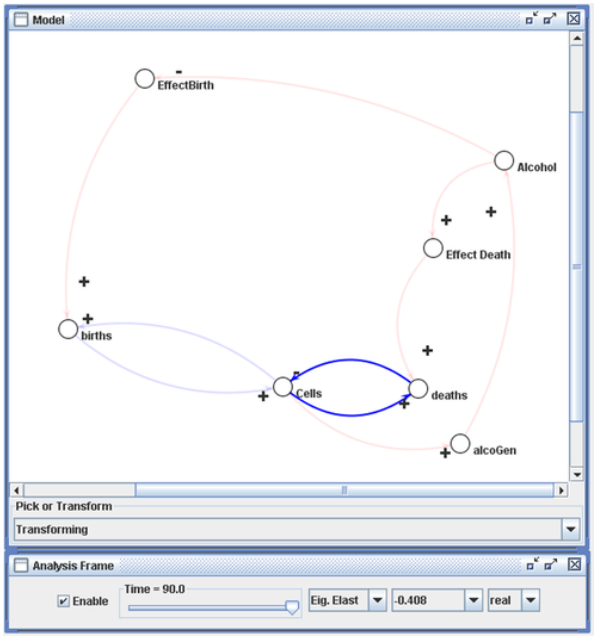
(a) Elasticities at $t = 0$, for λ_1



(b) Elasticities at $t = 37$, for λ_1

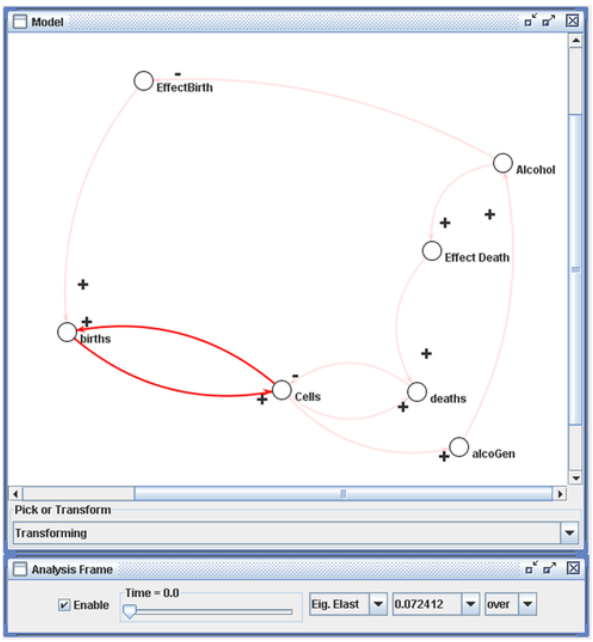


(c) Elasticities at $t = 80$, for λ_1

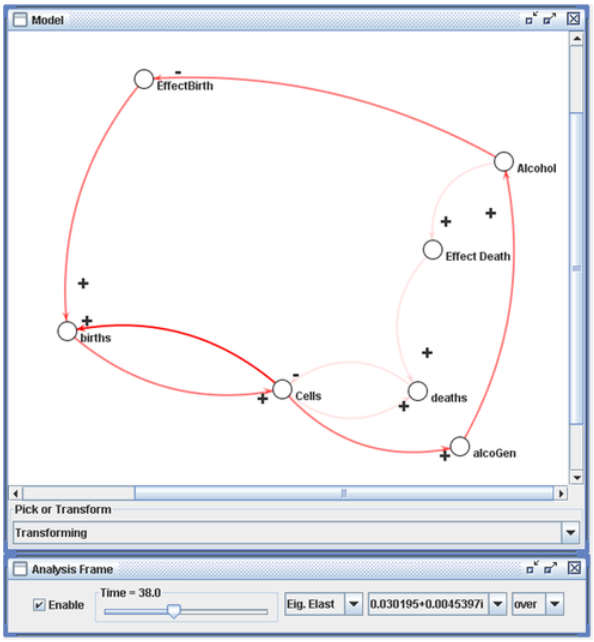


(d) Elasticities at $t = 90$, for λ_1

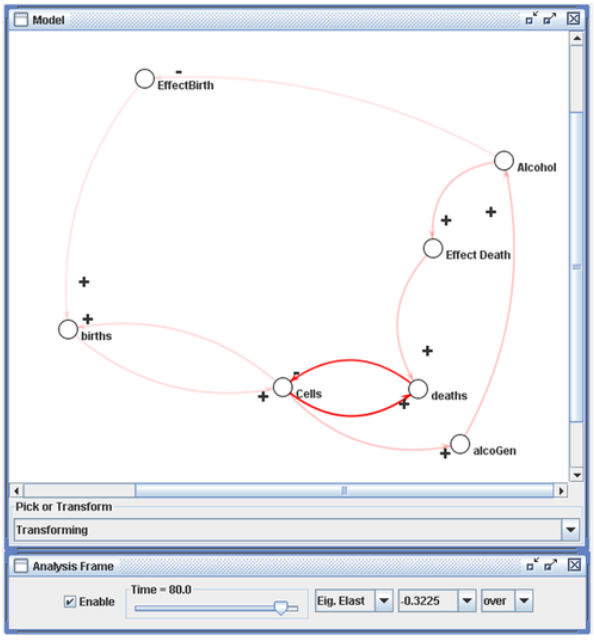
Figure 7: The development in time of real eigenvalue elasticities to a particular eigenvalue in a simple model. The opacity and color of the links show the relative size and direction of influence at the selected moment in time on the selected eigenvalue. Note the transitions of influence from the minor loop between births and cells, to the major loops and finally to the minor loop between Cells and deaths. These influences would be constant in a linear model.



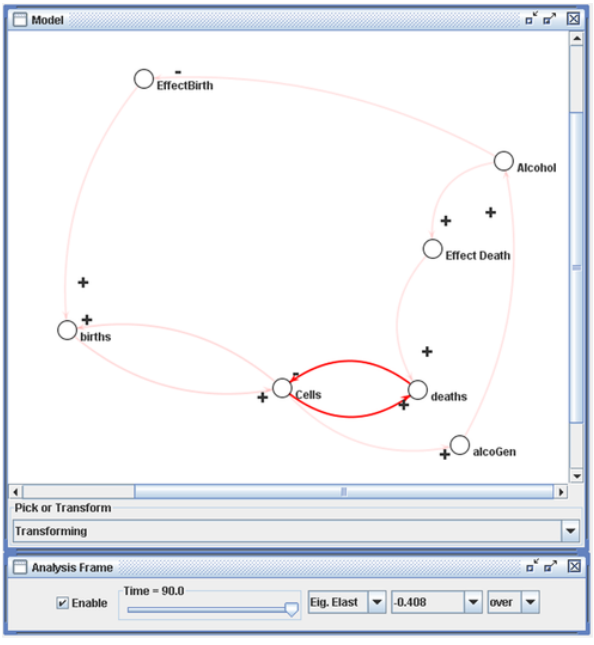
(a) Overall elasticities at $t = 0$, for λ_1



(b) Overall elasticities at $t = 38$, for λ_1



(c) Overall elasticities at $t = 80$, for λ_1



(d) Overall elasticities at $t = 90$, for λ_1

Figure 8: The development in time of overall eigenvalue elasticities to a particular eigenvalue in a simple model. The opacity of the links shows the relative size of influence at the selected moment in time on the selected eigenvalue.

5 Conclusions

The designed framework has been used to analyse a relatively large range of models. From small example models to medium sized classic models, such as the Market Growth (Morecroft, 1983; Forrester, 1968) model. In building the framework the algorithms of Güneralp (2006) have been generalised to be applicable, without modification, to higher order models.

In addition, the results of the analysis can be translated back to a type of diagram that forms an extension to causal diagrams. The visualisation relies on formal model analysis to highlight influential elements of structure. A dynamic causal diagram, changing over time, shows how the non-linearities in the model drive the waxing and waning of the influence of different sets of loops. This visualizes how the structure of the system generates its behaviour, in terms familiar to modellers.

This represents a significant departure from only ascribing dominance to a particular loop. Instead of just displaying the most influential loop(s) this also displays the elements of structure that work against the most dominant elements. For instance, in the Yeast model, it not only shows those loops directly responsible for growth during the first phase of the model, but also those loops that restrain this growth.

It must be noted that the techniques used in generating these images are still a topic of research themselves. Several remaining challenges can be identified:

- The further development of eigenvector based methods (Güneralp, 2006; Saleh et al., 2006). Recent work has addressed some of the problems of eigenvalue based analysis by incorporating eigenvectors into the methods, but issues still remain (Güneralp, 2006; Saleh et al., 2006).
- The link from the results of the formal analyses to the modelling process. How can formal model analysis be of use in the different phases of the modelling process? For instance, designing alternatives, simplifying the model, or evaluating system boundaries.
- The generation of system stories based on the outcome formal model analyses. The application of the currently available analyses are still the domain of experts familiar with the methods. However, if the methods are to be of any use, their outcome will have to be translated into language understandable to the client.

References

Forrester, J. W. (1968). Market growth as influenced by capital investment. *Industrial Management Review (MIT)*, 9(2). 8, 16

Forrester, J. W. (1987). Nonlinearity in high-order models of social systems. *European Journal of Operational Research*, 30:104–109. 1, 12

Forrester, N. (1982). *A Dynamic Synthesis of Basic Macroeconomic Theory: Implications for Stabilization Policy Analysis*. PhD dissertation, MIT, Cambridge, MA. 2, 6

Güneralp, B. (2005). Towards coherent loop dominance analysis: Progress in eigenvalue elasticity analysis. In *Proceedings of the 23rd International Conference of the System Dynamics Society*, Boston. System Dynamics Society, Albany, NY. 11

Güneralp, B. (2006). Towards coherent loop dominance analysis: progress in eigenvalue elasticity analysis. *System Dynamics Review*, 22:263–289. 2, 3, 6, 16

Güneralp, B. and Gertner, G. (2006). Feedback loop dominance analysis of two tree mortality models: Relationship between structure and behavior. *Tree Physiology*. In Review. 6

Jacobs, P. H. M. (2005). *The DSOL Simulation Suite: Enabling Multi-formalism Simulation in a Distributed Context*. PhD dissertation, Delft University of Technology. 4

Kamp-mann, C. E. (1996). Feedback loop gains and system behaviour. In *Proceedings of the 1996 International System Dynamics Conference*, Boston. System Dynamics Society, Albany, NY. 3, 4, 6, 8

Kampmann, C. E. (1996). Feedback loop gains and system behaviour. Unpublished. 2

Kampmann, C. E. and Oliva, R. (2006). Loop eigenvalue elasticity analysis: three case studies. *System Dynamics Review*. 2, 11

Mojtahedzadeh, M. T., Andersen, D., and Richardson, G. P. (2004). Using digest to implement the pathway participation method for detecting influential system structure. *System Dynamics Review*, 20(1):1–20. 1

Morecroft, J. D. W. (1983). System dynamics: Portraying bounded rationality. *Omega*, 11:131–142. 8, 16

North, M. J., Collier, N. T., and Vos, J. R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16. 4

Oliva, R. (2004). Model structure analysis through graph theory: partition heuristics and feedback structure decomposition. *System Dynamics Review*, 20(4):313–336. 3, 4, 6

Phaff, H. W. G. (2006). Investigating model behavioural analysis: A critical examination of two methods. In *Proceedings of the 2006 International System Dynamics Conference*, Nijmegen. System Dynamics Society. 11

Richardson, G. P. (1984). *The Evolution of the Feedback Concept in American Social Science*. PhD dissertation, school = "MIT, Cambridge, MA",. 1, 12

Richardson, G. P. (1995). Loop polarity, loop dominance, and the concept of dominant polarity. *System Dynamics Review*, 11(1):67–88. 8

Richardson, G. P. (1996). Problems for the future of system dynamics. *System Dynamics Review*, 12(2):141–157. 1

Saleh, M. (2002). *The Characterization of Model Behavior and its Causal Foundation*. PhD dissertation, Department of Information Science, University of Bergen. 11

Saleh, M., Oliva, R., Kampmann, C. E., and Davidsen, P. (2006). Eigenvalue analysis of system dynamics models: Another perspective. In *Proceedings of the 2006 International System Dynamics Conference*, Nijmegen. System Dynamics Society. 16

Sterman, J. D. (2000). *Business Dynamics. Systems Thinking and Modeling for a Complex World*. Irwin McGraw-Hill, Boston, MA. 1

Wolstenholme, E. F. and Coyle, R. G. (1983). The development of system dynamics as a methodology for system description and qualitative analysis. *Journal of the Operational Research Society*, 34:569–581. 1

Appendices

A Sensitivity Analysis

The following snippet of code shows how to do a full sensitivity analysis on any model, with the trajectories of the model with each parameter perturbed as output.

```
function trajectories = sensitivityData(anyModel, integrator, timespan,...
    parameters, relPerturbation)
    % for every parameter in the selected parameterset, run sensitivity
    % analyses. Needs simpleinteg, a wrapper function around the model.

    global model;
    model = anyModel;
    initials = anyModel.getStateValues();

    function runResult = sensitivity(parameter)
       defValue = parameter.getValue;
       parameter.setValue(defValue + defValue*perturbation);
       [t, y] = integrator(@simpleInteg, timespan, initials);
       parameter.setValue(defValue);
       runResult = {y};
    end

    % Sensitivity maps parameters to trajectories
    % Perturb positively
    trajpos = cellfun(@sensitivity,...
        cell(parameters), 'UniformOutput', false);

    perturbation = -relPerturbation;
    % Perturb the other way
    trajneg = cellfun(@sensitivity,...
        cell(parameters), 'UniformOutput', false);
    trajectories = [trajpos; trajneg];
end
```

The following snippet does the same for an active nonlinear test.

```
function opt = activeNonLinearTest()
    % Maximize the absolute difference for the selected state
    % A 10% deviation in parameter values is allowed.

    % set up the model
    import nl.tudelft.tpm.pa.sd.yeast.proxy.*
    yeast = Yeast();
    yeast.constructModel;
    global model;
    model = yeast;
    initials = yeast.getStateValues();

    % get the reference run
    [t, refrun] = ode45(@simpleInteg, 0:1:90, initials);

    parsJ = yeast.getParameters();
    refParValues = cellfun(@(par)par.getValue(),...
        cell(parsJ.toArray()));
```

```
19      numPars = size(refParValues, 1);
20      lowerbound = 0.9*ones(size(refParValues));
21      upperbound = 1.1*ones(size(refParValues));
22
23      function metric = fitnessWrapper(parameterVector)
24          % For loop should not be in here
25          for idx = 0:numPars-1
26              parsJ.get(idx).setValue(refParValues(idx+1).*...
27                  parameterVector(idx+1));
28          end
29          [t, mod_run] = ode45(@simpleInteg, 0:1:90, initials);
30          % Take the absolutely largest difference in number of cells
31          % during the run as a metric for deviation. Bad metric, but ok for
32          % demonstration purposes.
33          % Function seeks to minimize this, so divide
34          metric = 1/(max(abs(mod_run(:,1) - refrun(:,1))));
35      end
36
37      % Returns parameter settings for max deviation
38      x = ga(@fitnessWrapper,numPars,[],[],[],[],lowerbound,upperbound);
39      % Set parameter values to found maximum, plot
40      for index = 0:numPars-1
41          parsJ.get(index).setValue(refParValues(index+1).*x(index+1));
42      end
43      [t, maxdefrun] = ode45(@simpleInteg, 0:1:90, initials);
44      plot(t, refrun(:,1), t, maxdefrun(:,1));
45      opt = x;
46  end
```