

INTEGRATED, PATTERN-ORIENTED BEHAVIOR ANALYSIS AND TESTING
SOFTWARE (BATS) FOR SYSTEM DYNAMICS MODELS

by

Can Sücüllü

B.S., Industrial Engineering, Boğaziçi University, 2009

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering
Boğaziçi University

2014

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis advisor Gönenc Yücel. He literally carried me over through a successful thesis journey. Without his clear thinking and guidance, it would be impossible for me to complete this study. It has always been a privilege to work with him.

Secondly, I would like to thank my thesis committee member and “honorary” advisor Yaman Barlas. He has been a teacher of mine not only in industrial and systems engineering, system dynamics, systems thinking etc. but also in humanities, politics, philosophy and humor.

I would like to thank Ali Kerem Saysel for participating in my thesis committee and sharing his valuable comments.

My biggest thanks goes to Aynur & Alpay Sücüllü, my beloved mother and father, whose love and support I feel in every moment, past, present and future, and in every dimension of my life. I also would like to thank my brother Kutay and my lovely grandparents.

There are many others to thank; my colleagues at SESDYN Oylum Şeker, Nisa Önsel, Onur Özgün; my friends at the flamenco music community in İstanbul; and my friends in my improvisational theater group. Without art, life would be meaningless.

My special thanks goes to Sinem for being there with her precious endless support.

Last but not least, I believe the world is changing, humanity is changing, and in a “good” direction. One of the causes of this shift and also one of the effects of it is the free and fast sharing of knowledge and communication. I thank and respect all who participates in this new world.

ABSTRACT

INTEGRATED, PATTERN-ORIENTED BEHAVIOR ANALYSIS AND TESTING SOFTWARE (BATS) FOR SYSTEM DYNAMICS MODELS

Analysis of model behavior is mainly conducted in a pattern-based manner in system dynamics (SD) methodology. In pattern-based evaluation of model outputs, similarity of the overall behavior pattern (e.g. growth-and-decline, S-shaped growth, oscillations) and of specific pattern characteristics (e.g. inflection points, stabilization level, periods, amplitudes) are more important than point-by-point similarity measures such as sum of squared errors. Although some output analysis tools that address this special pattern focus are available, these tools and methods lack usability and are fragmented. In this study, a new analysis environment is proposed, and this environment is developed into an integrated standalone software. This software, namely Behavior Analysis and Testing Software (BATS), integrates a behavior pattern classification algorithm and a selection of statistical methods for analysis of steady-state behavior patterns. Apart from enabling comparison of dynamical patterns with these algorithms and methods, BATS includes structured processes that enable the user to conduct automated hypothesis testing, behavior space exploration, and sensitivity analysis. In its current state, BATS can seamlessly communicate with SD modeling software (e.g. Vensim) as well as other common data sources. The thesis also provides illustrative examples of how BATS can assist the modeler and/or analyst in various tasks in several stages of modeling; model credibility by indirect structure testing, output evaluation by behavior pattern tests, behavior sensitivity analysis, model calibration, policy analysis and design. Considering its pattern focus, user-friendly interface, communication capabilities with data sources, and automation capabilities, BATS can be an important contribution to the analysis toolset of SD methodology.

ÖZET

SİSTEM DİNAMİĞİ MODELLERİ İÇİN ENTEGRE, ÖRÜNTÜ-ODAKLI DAVRANIŞ ANALİZİ VE TESTİ YAZILIMI (BATS)

Sistem dinamiği yönteminde, model davranışlarının analizi temelde örüntü eksenli yapılmaktadır. Model çıktılarının örüntü eksenli değerlendirilmesinde, davranışın genel biçimi (ör. artışı takip eden azalma, S-biçimli büyüme, salınımlar) ve belirli örüntü özellikleri (ör. bükülme noktaları, karar seviyesi, dönemler, genlikler) karesel yanılı toplamı gibi noktasal bazlı benzerlik ölçütlerinden daha çok önem arz etmektedir. Bu özel durumu ele alan bir takım çıktı analiz araçları önceden geliştirilmiştir, fakat bu araç ve yöntemlerin dağınık ve kullanılabilirliklerinin düşük oldukları tespit edilmektedir. Bu çalışmada yeni bir analiz ortamı önerilmekte ve bu ortam tek başına bir yazılım olarak bütünleştirilmektedir. Bu yazılım, Davranış Analizi ve Testi Yazılımı (BATS), bir davranış örüntüsü sınıflandırma algoritması ve kararlı durum davranışlarının analizi için seçilmiş bir grup istatistiksel yöntemleri bütünleştirmektedir. Mevcut algoritma ve yöntemleri kullanarak dinamik örüntülerin karşılaştırılmasına imkan tanımanın yanında, BATS kullanıcının otomatik hipotez testi, davranış uzayı keşfi ve hassasiyet analizi yapmasına imkan veren düzenli süreçler içermektedir. Mevcut durumunda BATS sistem dinamiği modelleme yazılımlarıyla (ör. Vensim) ve yaygın veri kaynaklarıyla sorunsuz bir şekilde iletişim kurabilmektedir. Ayrıca bu tezde, BATS'ın modelleme sürecinde karşılaşılan dolaylı yapısal testlerle model geçerliliğinin sınanması, davranış testleri, davranış hassasiyet analizi, model kalibrasyonu, politika analizi ve tasarımı aşamalarında modellemeci ve/veya araştırmacıya nasıl yardımcı olabileceği açıklayıcı örneklerle gösterilmektedir. Örüntü odaklı içeriği, kullanıcı dostu arayüzü, veri kaynaklarıyla iletişim ve otomatik analiz yapma olanakları dikkate alındığında BATS, sistem dinamiği yönteminin analiz setine önemli bir katkı olarak öne sürülmektedir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	ix
LIST OF TABLES	xvi
LIST OF SYMBOLS	xvii
LIST OF ACRONYMS/ABBREVIATIONS	xviii
1. INTRODUCTION	1
2. PROBLEM DEFINITION AND METHODOLOGY	4
2.1. Problem Statement	4
2.2. Research Objectives	6
2.2.1. Pattern-Oriented Behavior Analysis	7
2.2.2. Automation	8
2.2.3. Communication with the Existing Modeling Software	9
2.2.4. User-friendly Interaction	11
2.2.5. Discussion	12
2.3. Research Procedure	14
3. BACKGROUND INFORMATION ON PATTERN-ORIENTED BEHAVIOR ANALYSIS	16
3.1. Basic Properties of Dynamic Behavior	16
3.2. Literature Review	18
3.3. Discussion	21
4. SELECTED METHODS FOR DYNAMIC BEHAVIOR ANALYSIS	22
4.1. Preprocessing of Dynamic Behavior	22
4.1.1. Separation of Transient and Steady-State Behavior	23
4.1.2. Smoothing	23
4.2. Classification of Dynamic Behavior Patterns Based on ISTS Algorithm	24
4.3. Multi-Step Procedure of Behavior Testing Software (BTS)	30
4.3.1. Trend Regression and Removal	30

4.3.2.	Autocorrelation	31
4.3.3.	Spectral Density Function	33
4.3.4.	Amplitude Estimation	34
4.3.5.	Cross Correlation	35
4.3.6.	Coefficients of Similarity	36
4.4.	Graphical Inspection	37
4.5.	Summary of Behavior Analysis Methods and Procedures	38
5.	DEVELOPMENT AND PROPERTIES OF BATS	40
5.1.	Programming Language Selection	41
5.2.	Communication with Modeling Software	42
5.3.	Main Design Elements	43
5.3.1.	Graphical User Interface	44
5.3.2.	Workspace and DataItems	47
5.3.3.	Plotbook and Tabs	48
5.4.	Data Inputting and Preparation Features	50
5.4.1.	Load Data From File	50
5.4.2.	Draw	51
5.4.3.	Split	53
5.4.4.	Crop	54
5.4.5.	Exponential Smoothing	54
5.4.6.	Moving Average	55
5.5.	Data Analysis Features	55
5.5.1.	Plot	57
5.5.2.	Trend	59
5.5.3.	Autocorrelation	60
5.5.4.	Spectral Density	61
5.5.5.	Amplitude Estimation - Constant	61
5.5.6.	Cross Correlation	64
5.5.7.	Graphical Comparison	65
5.5.8.	Classify	66
5.6.	Model Related Features	67

5.6.1.	Model Docking Window	67
5.6.2.	Hypothesis Tester	71
5.6.3.	Behavior Space Classifier	73
5.6.4.	Behavior Class Mapper	74
6.	DEMONSTRATION AND USAGE MODES OF BATS	76
6.1.	Extreme Condition Testing	76
6.1.1.	Test Model: Density-Dependent Growth Model	76
6.1.2.	General Usage Procedure	77
6.1.3.	Tests and Results	79
6.2.	Behavior Pattern Tests	83
6.3.	Sensitivity Analysis	95
6.3.1.	Test Model: Temperature Adjustment Model	95
6.3.2.	Tests and Results	95
6.4.	Parameter Calibration	100
6.5.	Policy Design	102
6.6.	Policy Analysis	104
6.6.1.	Test Model: Predator-Prey Model	104
6.6.2.	Policies and Results	104
6.7.	Summary of Demonstration and Usage Modes Chapter	114
7.	CONCLUSION	115
	APPENDIX A: ESSENTIAL CODE PARTS	119
A.1.	ISTS Algorithm	119
	APPENDIX B: EQUATIONS OF TEST MODELS	127
B.1.	Density Dependent Growth Model	127
B.2.	Temperature Adjustment Model	127
B.3.	Predator-Prey Model	128
	REFERENCES	129

LIST OF FIGURES

Figure 2.1.	Automation levels.	8
Figure 2.2.	Four requirements for an ideal model analysis environment.	13
Figure 3.1.	Illustrative example for showing weaknesses in point based metrics.	18
Figure 4.1.	ISTS Algorithm.	25
Figure 4.2.	Generic dynamic patterns that can be classified by ISTS Algorithm (Kantar, 1999).	26
Figure 4.3.	Dynamic behavior pattern examples.	28
Figure 4.4.	Autocorrelation functions for random, oscillatory and non-stationary data with trend.	32
Figure 4.5.	Period finding algorithm based on autocorrelation function.	33
Figure 4.6.	Spectral densities of a 2-period oscillation.	34
Figure 4.7.	Cross correlation plots for two pairs of signals.	36
Figure 5.1.	Main screen of BATS.	40
Figure 5.2.	Overview of Elements of BATS.	43
Figure 5.3.	SDI design of SiS (Boğ and Barlas, 2005) and MDI design of BTS (Barlas <i>et al.</i> , 1997).	45

Figure 5.4.	Representative customized views of BATS.	46
Figure 5.5.	The class diagram for <i>Workspace</i> and <i>DataItem</i>	47
Figure 5.6.	The appearance of <i>Workspace</i> after a hypothetical analysis session.	48
Figure 5.7.	The class diagram for <i>Plotbook</i> and <i>Tab</i>	49
Figure 5.8.	<i>Data</i> menu for data inputting and preparation features.	50
Figure 5.9.	Input dialog for <i>Draw</i> tool.	51
Figure 5.10.	Illustration of <i>Draw</i> feature.	52
Figure 5.11.	Screenshot for <i>Split</i> feature.	53
Figure 5.12.	Screenshot for <i>Crop</i> feature.	54
Figure 5.13.	Screenshot for <i>Exponential Smoothing</i> feature.	55
Figure 5.14.	Screenshot for <i>Moving Average</i> feature.	56
Figure 5.15.	<i>Analysis</i> menu for data analysis features.	57
Figure 5.16.	Screenshot for <i>Plot</i> feature.	58
Figure 5.17.	Screenshot for <i>Plot Multi</i> feature.	58
Figure 5.18.	Screenshot for <i>Trend</i> feature.	59
Figure 5.19.	Screenshot for <i>Autocorrelation</i> feature.	60

Figure 5.20. Screenshot for <i>Spectral Density</i> feature.	61
Figure 5.21. Screenshot for <i>Amplitude Estimation - Constant</i> feature.	62
Figure 5.22. Screenshot for <i>Amplitude Estimation Trend</i> feature.	63
Figure 5.23. Screenshot for <i>Cross Correlation</i> feature.	64
Figure 5.24. Screenshot for <i>Graphical Comparison</i> feature.	66
Figure 5.25. <i>Model</i> menu for model analysis features.	67
Figure 5.26. Screenshot for <i>Model Docking Window</i> feature.	68
Figure 5.27. Two alternatives of <i>Run Specifications</i> dialog box.	69
Figure 5.28. Hypothesis testing specifications dialog box.	71
Figure 5.29. <i>Hypothesis Tester</i> results screen.	72
Figure 5.30. Settings dialog box for <i>Behavior Space Classifier</i>	73
Figure 5.31. Screenshot for <i>Behavior Space Classifier</i> results.	74
Figure 5.32. Settings dialog box for <i>Behavior Class Mapper</i>	75
Figure 5.33. Screenshot for <i>Behavior Class Mapper</i> (results for s-shaped growth pattern).	75
Figure 6.1. Stock-flow diagram of the density-dependent growth model.	77

Figure 6.2.	Preparation of the model for <i>Hypothesis Tester</i> feature.	78
Figure 6.3.	Test specifications for the first hypothesis.	79
Figure 6.4.	Test results for the first hypothesis.	79
Figure 6.5.	Test specifications for the second hypothesis.	80
Figure 6.6.	Test results for the second hypothesis.	80
Figure 6.7.	Test specifications for the third hypothesis.	81
Figure 6.8.	Test results for the third hypothesis.	81
Figure 6.9.	Screens from the end of the analysis session.	82
Figure 6.10.	Plot for actual and model.	84
Figure 6.11.	Split for actual.	84
Figure 6.12.	Split for model.	85
Figure 6.13.	Plot for steady-state parts of actual and model after the splitting operation.	85
Figure 6.14.	Trend regression results for actual.	86
Figure 6.15.	Trend regression results for model.	87
Figure 6.16.	Autocorrelation function results for actual.	87

Figure 6.17. Autocorrelation function results for model.	88
Figure 6.18. Spectral densities of actual.	88
Figure 6.19. Spectral densities of model.	89
Figure 6.20. Autocorrelation test results for actual and model.	90
Figure 6.21. Statistics summary results for actual and model.	90
Figure 6.22. Amplitude estimation results for actual.	91
Figure 6.23. Amplitude estimation results for model.	91
Figure 6.24. Crosscorrelation function results for actual and model.	92
Figure 6.25. Graphical comparison results for actual and model.	93
Figure 6.26. Stock-flow diagram of the temperature adjustment model.	96
Figure 6.27. Settings for maximum likelihood class search.	96
Figure 6.28. <i>Behavior Space Classifier</i> results. Different views in bottom plot are obtained by clicking on the top grid.	98
Figure 6.29. <i>Behavior Space Classifier</i> results continued.	99
Figure 6.30. Settings for <i>sshgr</i> likelihood search.	100
Figure 6.31. Results for <i>sshgr</i> likelihood search.	101

Figure 6.32. Settings for <i>nexgr</i> likelihood search.	102
Figure 6.33. Results for <i>nexgr</i> likelihood search.	103
Figure 6.34. Stock-flow diagram of the predator-prey model.	105
Figure 6.35. Run specifications for base policy.	106
Figure 6.36. Simulation results for base policy.	106
Figure 6.37. Run specifications for policy 1.	107
Figure 6.38. Simulation results for policy 1.	107
Figure 6.39. Run specifications for policy 2.	108
Figure 6.40. Simulation results for policy 2.	108
Figure 6.41. Run specifications for policy 3.	109
Figure 6.42. Simulation results for policy 3.	109
Figure 6.43. Workspace after exporting all of the simulation results.	110
Figure 6.44. Amplitude estimation for base policy, (<i>Amp.</i> = 53).	111
Figure 6.45. Amplitude estimation for base policy, (<i>Amp.</i> = 112).	111
Figure 6.46. Amplitude estimation for base policy, (<i>Amp.</i> = 62).	112
Figure 6.47. Amplitude estimation for base policy, (<i>Amp.</i> = 137).	112

Figure 6.48. Behavior of X (prey population) for all policies.	113
--	-----

LIST OF TABLES

Table 2.1.	Stages of system dynamics methodology that involve output evaluation.	5
Table 4.1.	Description of generic dynamic patterns in Figure 4.2 (Kanar, 1999).	27
Table 4.2.	ISTS Algorithm results for behaviors in Figure 4.3.	29
Table 4.3.	Multi-Step procedure by Barlas.	30
Table 4.4.	Trend fitting and removal formulas	31
Table 6.1.	Summary table for behavior validity pattern tests.	94
Table 6.2.	Parametric representations of policy alternatives regarding the predator-prey model.	105
Table 6.3.	Analysis of model outputs for the policy alternatives.	113
Table 6.4.	Summary of features used in this chapter.	114

LIST OF SYMBOLS

$Cov(k)$	Autocovariance function at lag k
k	Lag index
$r(k)$	Autocorrelation function at lag k
t	Time index
α	Exponential smoothing coefficient
β	Moving average terms

LIST OF ACRONYMS/ABBREVIATIONS

Amp.	Amplitude
BATS	Behavior Analysis and Testing Software
BTS	Behavior Testing Software
BTS II	Behavior Testing Software II
Diff.	Difference
Eql.	Equilibrium
ISTS	Indirect Structure Testing Software
MSE	Mean squared error
POPS	Pattern Oriented Parameter Specifier
SiS	Validity Design and Parameter Calibration Software

1. INTRODUCTION

System dynamics is an approach to deal with systemic real world problems. It is useful for long-term policy problems that necessitate continuous managerial action and control. Also these problems should be of systemic and feedback nature. Main objective of a system dynamics study is to understand the underlying structure of a dynamic problem, and then design and test policies that reduce or eliminate it.

A typical system dynamics study starts with selection of an appropriate problem. System dynamics is helpful for certain types of real world problems. The problem should be a dynamic management and control problem, should involve feedbacks between its different variables and managerial action, and should emerge as a result of the endogenous structure of the system, not due to the external forces. The problem identification stage is then followed by formulation of a dynamic hypothesis and construction of the mental model of the system with respect to the problem. Mental models are unable to be tested and used as a means of communication as they are unstructured and subjective. They become beneficial when transferred into formal models. Formal models can occasionally be in the form of a set of mathematical equations, but in real practice they are in the form of computer simulation models due to the complexity issues. Thus, simulation is an essential part of the methodology. Model construction is followed by model validation phase, in which the usefulness of the model with respect to its purpose is assessed and improved. Model analysis is defined as understanding why and how the model behaves the way it does. In practice model analysis is done by making use of sensitivity analysis, which is essentially evaluating changes in model behavior in response to changes in its parameters and/or structures. The model that is tested and analyzed can be used for policy evaluation and design. Finally, the modeling study ends with the policy recommendations and other concluding knowledge and insights (Barlas, 2002).

There are two important concepts related to a formal simulation model; structure and dynamic behavior. Structure is the totality of the relationships that exist between

the system variables, and dynamic behavior is the performance patterns of the variables created by the operation of the structure of the system over time (Barlas, 2002). The structure of a system dynamics model consists of the set of relations between model variables, mathematically represented in the set of differential and/or difference equations. The dynamic behavior of the model is the output that is generated by the operation of model variables over simulated time. Thus stated, the focus of this thesis study is related to the dynamic model behavior.

Dynamic behavior analysis is encountered in various stages of a modeling study. In model creation phase, it is important for parameter estimation and calibration of the model according to the historical data or some desired behavior. The crucial phase of model validation can be discussed in two categories; structural validity and behavior pattern validity. In structural validity there are certain tests that involve simulation such as extreme condition tests, in which comparing the model performance with the hypothesized real behavior is essential. In behavior pattern validity, the similarity of pattern related characteristics such as trends, minima, maxima, periods, amplitudes between model output and real life data is important. In sensitivity analysis, there is a need for categorization of large numbers of model outputs with respect to certain rules and purposes such as finding the parameter range that eliminates oscillations. Finally, in policy analysis and design phase the outputs are ranked according to policy objectives so that the most robust one under alternative scenarios can be chosen or designed.

Owing to the long-term policy orientation of system dynamics modeling the task of dynamic behavior analysis or model output evaluation necessitates special care. It should be done in a pattern-oriented manner, that is, the primary concern is on pattern related measures. For example, whether the behavior is an s-shaped-growth pattern or not is more important than the exact value at a certain point in time. This pattern-orientation necessitates construction of formal analysis methods. A literature review on available model analysis methods/tools and real life modeling projects reveals that formal analysis methods are utilized at a much less than desired or deserved level in practice. This results in subjective analysis efforts and results, and even lack of applied

formal analysis processes.

The second issue related to model behavior analysis is encountered at a task level. For a specific task such as sensitivity analysis, the number of simulation experiments needed grow exponentially with respect to parameter and structural changes. As a consequence of this time-consuming nature of the whole process, the number of simulation experiments actually performed is usually much lower than the amount necessary to conduct extensive survey on model behavior.

2. PROBLEM DEFINITION AND METHODOLOGY

2.1. Problem Statement

System dynamics methodology has been criticized for its lack of quantitative analysis methods. While these criticisms have been partially addressed by several papers, there is still a need for further formalization of the validity testing and behavior analysis procedures of the methodology. This need has been mentioned by several authors (Richardson, 1996; Barlas, 1996, 2007; Sterman, 2000). In this study the focus is on model testing and analysis efforts where evaluation of dynamic model behavior is involved.

Dynamic behavior analysis is a significant task in system dynamics methodology. It is encountered in various stages of a modeling study. The four major steps where output evaluation is critical are listed below.

- Model Calibration and Parameterization
- Model Validation and Testing
- Sensitivity Analysis
- Policy Analysis and Design

A more specific information regarding these steps with purposes and representative questions is given in Table 2.1. It can be seen that output evaluation serves for different purposes in different steps of the methodology.

Being a descriptive and causal policy modeling approach, output evaluation in system dynamics have special characteristics differing from statistical forecasting models. Assessing similarity of dynamic behaviors should be done by putting more emphasis on pattern characteristics rather than point-by-point proximity. For model validation and calibration purposes, reproduction of pattern characteristics like trends, means, variances, periodicities, amplitudes are more important than point precision. Likewise,

Table 2.1. Stages of system dynamics methodology that involve output evaluation.

Task	Purpose	Example Question
Model Calibration & Parameterization	Fine tune the model equations/parameters to match/fit the historical time-series	What is the best value of my parameter x in order my model to best fit the historical behavior?
Model Validation and Testing	Gain confidence on model structure and its behavior mode by passing various validation tests	Does the model generate similar results with the expected behavior of the real system under the same extreme conditions?
Behavior Sensitivity Analysis	Identify different dynamic pattern modes and behavior sensitivity of the model	What kinds of behavior modes does the model generate within the feasible range of parameters?
Policy Analysis and Design	Design and test alternative policies under various scenarios	Which policy, represented by a set of parameters, yields the most robust behavior?

for policy analysis purposes, pattern prediction is more important than point prediction. Hence, output evaluation should be done in a pattern-oriented manner. This pattern orientation brings subjectivity to output evaluation by rendering standard statistical measures inadequate.

As a result, output evaluation is predominantly exercised subjectively. It may even be claimed that subjective nature of behavior analysis is one of the weakest aspects of the system dynamics methodology. A literature review on published modeling studies reveals that conclusions related to dynamic behavior are roughly similar to statements like “this behavior looks similar to that one” (e.g. Arenas, 2005; Otto and Simon, 2008). Another conclusion of this review is that a statistical metric called Theil statistics (Sterman, 1984) is the most widely and in certain cases the only analysis measure utilized (e.g. Eskinasi *et al.*, 2009; Ghaffarzadegan and Tajrishi, 2010). These two types of conclusions are acceptable and valid; however, they are more open to subjective criticisms and are better be supported by formal objectively calculated indicators/metrics in order to be scientifically more grounded. Also the availability of

formal behavior analysis methods would trigger an increase in number and variety of tests that involve model behavior. Designing formal output evaluation methods and integrating them in a well-structured environment would also increase the effectiveness of performing tests and communicating results. This would for example, enable effectively finding the better policy, increase the number of extreme conditions tests conducted, improve knowledge of different behavior modes the model can exhibit. This would also improve the field's communication with the scientific and non-scientific world.

The second problem related to dynamic behavior analysis is the time-consuming nature of the overall process. A full search in the feasible ranges of the parameters results in various types of problems in addition to the time-consuming nature itself (Hearne, 1985; Drechsler, 1998; Miller, 1998; Yücel and Barlas, 2011). As a result of this problem, less experiments are conducted than the favorable number.

Third problem observed in dynamic behavior analysis is that the available analysis methods/tools/software are not extensively being used. In the model analysis literature there are several examples that promise effective behavior analysis procedures; however, they lack the proper user base.

In conclusion, the problems addressed in this thesis can be compactly stated as dynamic behavior analysis in system dynamics has subjective characteristics that should be formalized, and there exists no automated user friendly tools that enable performing integrated model behavior analysis.

2.2. Research Objectives

In the previous section, the three problems related to dynamic behavior analysis procedure and methods are identified as subjectivity in analyses, time-consuming nature of the process and lack of usage of the existing methods. Proposed solution to these problems is defined as designing an analysis environment where the modeler/-analyst can perform pattern-oriented behavior analysis based on relevant quantitative methods and techniques, at various steps of a modeling study/project. Of equal im-

portance is the usability issue of the proposed analysis environment. This objective is addressed with three aspects. In order to achieve higher usability of this new analysis environment, it should; (i) be able to offer automated analysis, (ii) be integrated with existing modeling software, and (iii) have user-friendly interaction capabilities. These three properties combined with the pattern-orientation principle make directions for the research objectives. They are described in detail in the following sections.

2.2.1. Pattern-Oriented Behavior Analysis

This is the analytical backbone of the proposed environment. Selection of proper methods and algorithms is a crucial task. This selection in a way forms the toolbox of the modeler. An ideal toolkit for model behavior analysis consists of practical analytical functions and methods that are mostly based on existing and widely accepted research. It is also desired that these methods are intuitive, easy to understand, implement and communicate. Finally, these methods should support possibilities for visualization of their results. Highly specialized analysis methods are regarded as extra elements that can be included at a later phase.

The pattern-orientation principle stems from the long term policy nature of the system dynamics methodology. In system dynamics characteristics of long term performance patterns are more important than exact values at single time points. Therefore in order to do similarity assessment and labeling on these pattern characteristics, special care should be taken. The pattern-orientation principle is discussed in detail in the following chapters, namely Chapter 3 and Chapter 4. Pattern-orientation is the basis for selection of formal analysis methods. When this condition is met, i.e. a proper collection of pattern-oriented analysis methods are collected, then it becomes possible to discuss about other requirements, namely automation, communication with modeling software and user-friendly interaction.

2.2.2. Automation

Automation is a requirement that is directed towards the time-consuming problem of overall dynamic behavior analysis task. Working with methods in an computerized and automated environment is beneficial in order to overcome this problem. “Automatedness” of an analysis method can be defined on different levels of complexity. In this study, automation levels will be categorized into three for simplicity; no-automation, functional-level automation and task-level automation. The discussion especially becomes more meaningful when there are lots of simulation runs are involved. At the lowest level, the operations are done manually on the model output. At the functional-level automation, we can speak of methods/tools that take model output as input, automatically calculate and report the results of that analysis mode, e.g. print autocorrelation results, plot spectral densities graph. At the task-level automation, we can speak of methods/tools that operate on the complete model structure. Figure 2.1 illustrates the different automation levels. In this thesis study, automation of analysis tools is desired to be achieved at least at a functional-level.

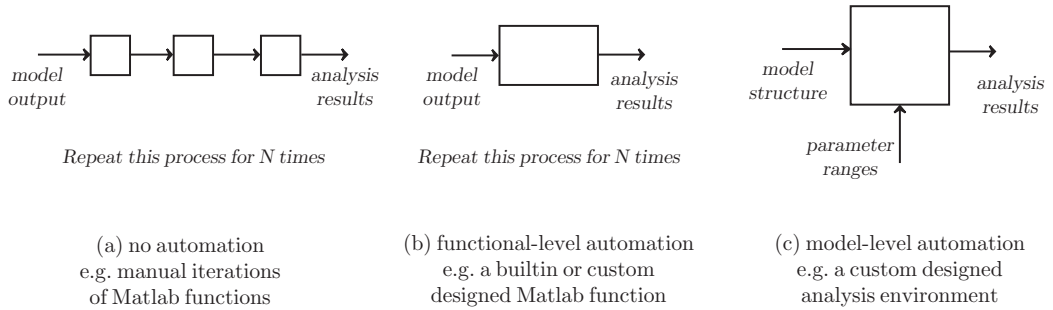


Figure 2.1. Automation levels.

It is relevant to discuss that among existing methods even the ones that enable task-level automated analysis are subject to the usability issue. There are beneficial examples in the literature that achieves task-level automated analysis, but lack the proper usage base due to this problem. For example; Yücel and Barlas (2011) address the time consuming combinatorial nature of parameter estimation and introduce a full automated (task-level) analysis tool called POPS. However, since 2011 there has been

no downloads and no usage demonstrations of this promising model analysis tool which is open source and available online¹. It can be stated as available or newly designed automated analysis methods are better be delivered with usability enhancements. The problem can be defined as in some cases using these methods requires familiarity with existing software, e.g. Matlab. A similar discussion can be made for model analysis tools based on Eigenvalue Elasticity Analysis (EEA) (Güneralp, 2006; Phaff, 2006). This method also offer beneficial analysis usage; however, fails to find the proper usage base it deserves.

There is clearly a need for carrying these kinds of methods/tools from the status of *proof of concepts* to the readily available methods/tools in the toolkit of the modelers.

2.2.3. Communication with the Existing Modeling Software

There are available specialized software packages for building dynamic simulation models. Some of the widely used ones are Stella, Vensim and Powersim. The literature, which introduces and describes a new method or procedure on model validation, sensitivity analysis, policy analysis, or parameter calibration concludes by stating that it would be beneficial to have that new method working in cooperation or integration with the existing modeling software (Barlas, 1996; Richardson, 1999; Ekşin, 2009; Yücel and Barlas, 2011). Current modeling software packages include internal analysis tools to some degree. For example, in Vensim Reality Check feature enables live, visual inspection of model behaviors as the parameters are manually modified serving as a manual sensitivity analysis tool (Peterson and Eberlein, 1994). Another example is SOPS (Krakenes, 2004; Moxnes, 2005), which is a stochastic policy optimization add-on to Powersim. Internal improvements of these commercial software as in this two examples are outside the topic of this thesis; however, it is possible to make external improvements by means provided with these software.

In the model analysis literature, there are examples that avoid the communication with the modeling environment step by transferring the model equations into a differ-

¹Available at <http://www.gyucel.net/research/POPS.html>

ent platform. For example, in the method called POPS, the new parameter calibration method is introduced by transferring the model structure equations to Matlab environment (Yücel and Barlas, 2011). Another example is Kampmann and Oliva (2006) in which loop eigenvalue elasticity analysis is implemented in Mathematica. There are only few model analysis applications where direct communication is used with the modeling software. They are Exploratory Modeling Analysis Workbench at TU Delft (EMA Team, 2011) which is an environment for policy analysis, and SiS (Boğ and Barlas, 2005) which is an indirect structure validity testing software enabling extreme condition tests.

In this study communication with one or more existing modeling software is proposed as one of the research objectives. This is suggested to improve usability of the methodology as opposed to the approach that requires the analyst to adopt a new analysis environment, e.g. creating a collection of Matlab functions. In order to motivate this decision the following facts are defined. The communication with existing modeling software is important due to the following issues:

- Adoption of a new analysis environment or a new programming language is troublesome for a significant portion of the existing practitioners. It is desired to reach as many of the audience as possible.
- Transfer of model equations from one platform to some other one is risky and open to transfer errors.
- Also some structures such as graph/table functions, delay formulations, conveyor structures, etc. are not easily transferable.
- It is desired to maintain and directly utilize the engine of the simulation (i.e. numerical integration methods and calculation order of model variables), and therefore benefit from the following fact: Since these software have been used and tested on many commercial and academic projects, there is enough confidence built on their numerical outcomes.

2.2.4. User-friendly Interaction

User-friendly human computer interaction is one of the defining characteristics of usability of the new analysis environment that is developed in this thesis study. It is concluded from the model analysis literature that, promising analysis methods should also be implemented in a user-friendly human computer interaction system in order them to be useful.

User-friendliness and the efficient human-computer interaction issue, with respect to the narrow scope of simulation modeling, is encountered in three areas. Firstly, it is encountered in actual model creation and building phase. This issue is related to modeling software packages such as Stella and Vensim. Secondly, it is encountered when simulation models are used in simulation gaming practices (e.g. management flight simulators, modeling for training). This issue is again related with existing simulation modeling software, and additionally related with online applications such as Forio. Thirdly and finally, it is encountered in model analysis software. For this item, two pioneer examples BTS & BTS II - Behavior Testing Software (Barlas *et al.*, 1997) and SiS - Validity Design and Parameter Calibration Software (Boğ and Barlas, 2005) can be given. While the first two mentioned issues are outside the scope of this study, the third one is highly relevant.

In simulation literature, there are no apparent examples where significant effort had been put on enhancing the user experience for the relevant user-friendliness issue mentioned above. Methodological model analysis tools are usually developed as a proof of concept. Due to various reasons, the tools that are introduced are not extensively developed and offered to other practitioners in an easily usable manner. An exceptional example can be given as BTS (Barlas *et al.*, 1997), which can be argued to have some level of user-friendliness. In this thesis study significant effort is given on creating an “actually usable software”.

In order to achieve a considerable level of usability, there are some desired properties. While they cannot be given as strict *must have* features, they will be considered

as requirements for achieving a satisfactory level of user-friendliness. These criteria are given in the following list. The sources for populating this list of criteria are from the literature on usability engineering (Nielsen, 1993; Johnson, 2008), design related open source web communities, accumulated know-how at the SESDYN Laboratory² at Boğaziçi University, and mostly from the personal experience of the author and the members of the thesis committee.

List of desired features that are suggested to increase level of usability is as follows. A user-friendly model behavior analysis software:

- Has graphical user interface rather than command line interface
- Is intuitive, easy to learn and use, and resembles everyday used software
- Compatible with commonly used software and file types
- Allows modern concepts like drag and dropping of external files
- Enables multiple views and working simultaneously with multiple processes
- Enables customizable views
- Can save/load session
- Has no major bugs, tested with a broad range of operations
- Cross-platform, works on different operating systems (Windows, Mac OS)
- Is extensible, i.e. developer friendly which is important for future work
- Can export quality graphics as images and print informative results as text
- Is distributed as a single packaged file, rather than multiple scattered files

2.2.5. Discussion

In summary, the issues discussed in this section on four research objectives can be pictured in Figure 2.2. Note that pattern-oriented tools are depicted in non-italics, rough places of the cited work are indicated on the diagram. Since pattern-orientation is the necessary requirement in model behavior analysis it is defined above these Venn diagram. In this figure the methods depicted in italic are not designed for pattern-oriented analysis purposes. The overall objective of this thesis is to build a model

²<http://www.ie.boun.edu.tr/labs/sesdyn>

behavior analysis software that meets the four requirements stated in this chapter. This can be represented by the intersection part in the figure, in which currently no analysis methods from the literature resides. It is relevant to note that these Venn diagrams are populated in a “has or does not have” basis, that is, detailed measures of pattern-orientation, automation, communication with modeling software and user-friendliness are not identified.

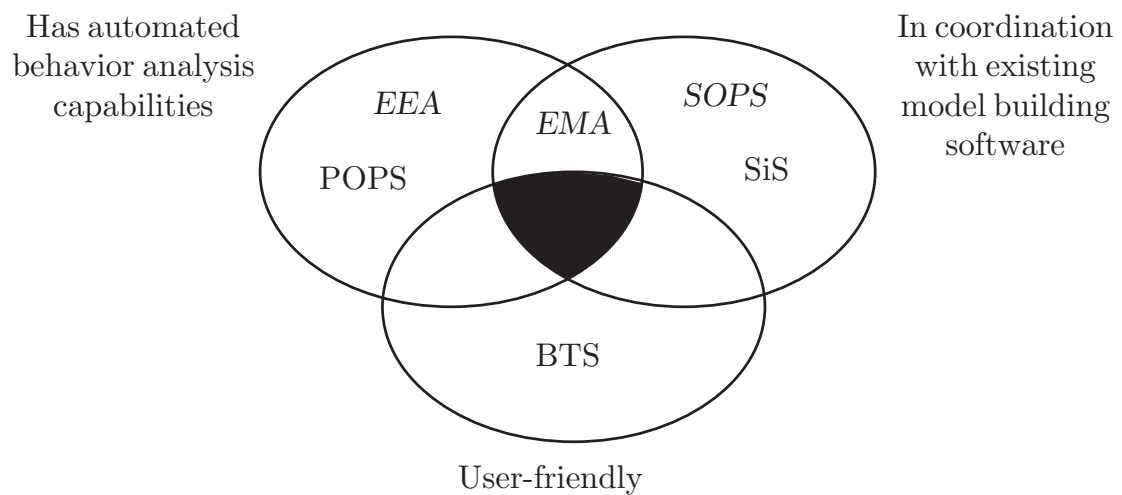


Figure 2.2. Four requirements for an ideal model analysis environment.

2.3. Research Procedure

In order to accomplish the research objectives proposed in the previous section, five steps are identified. This list presents the outline of the thesis study.

Firstly, literature review on available behavior analysis methods is given. In this step, initially answers to the questions “What kind of data is of interest in model behavior evaluation?”, “What are the different approaches between various disciplines?”, “What are the relevant ones to this study?” are investigated, and then the literature review is guided according to these answers and completed.

Secondly, selection of a toolbox for behavior analysis with extensions and improvements to the literature is completed. In this step pattern-oriented methods are procedures are gathered, and they as a whole comprise the analytical basis of the new environment.

Thirdly, implementation of the analysis toolbox as software is done. Issues like “Which programming language to utilize?”, “How should the software architecture be?”, “What are the input/output relations of functions?” and “How are the relationships of functions with each other?” are discussed in this part.

Fourthly, communication of existing modeling software, namely Vensim with the one developed at the previous step is done. In this step there are questions like “Which software will be implemented?” and “What are the capabilities that are desired?”.

Fifthly and lastly, elements for improving the user experience designed and implemented. In order to accomplish this step, the list of requirements mentioned in the previous section are used.

Organization of the thesis is connected to the above outline as such: Chapter 3 contains information related to the Step 1, namely literature review on pattern-oriented analysis methods. Chapter 4 contains information to Step 2, namely selection

and ingredients of the final analysis toolkit. Chapter 5 contains information related to Steps 3, 4 and 5, namely building blocks, essentials, architecture and features of the new environment. Also there is another chapter, Chapter 6 that demonstrates the usage modes of the new analysis environment.

3. BACKGROUND INFORMATION ON PATTERN-ORIENTED BEHAVIOR ANALYSIS

In the previous chapter, five research steps are defined in order to fulfill the research objectives of this thesis study. In this chapter, the first step, namely literature review on dynamic behavior analysis methods is covered. In the first section, important concepts related to model behavior analysis are discussed. Model behavior is defined as a typical time-series data, a collection of data points that are measured at regular time intervals over a time horizon. Other questions answered are: What is the objective of output evaluation? How is similarity of two time-series assessed according to the literature? How should it be properly assessed for policy oriented models? What are the comparison types? These questions serve as guidelines for reviewing the literature.

In the later section the complete literature review is given. Time-series data is analyzed in different ways in different disciplines for different purposes. The literature can roughly be categorized into two; statistical point based studies which treat time-series data as a collection of data points, and pattern recognition which treats time-series data as objects. Relevant points to system dynamics model analysis are discussed.

3.1. Basic Properties of Dynamic Behavior

Data analysis is a wide concept with applications to vast number of fields from medicine to astrophysics, and as the interest and ability in gathering data is increasing it will continue to grow in importance and precision (Shasha and Zhu, 2004; Huber, 2011). In the narrower context of simulation studies and particularly focusing on system dynamics simulation models, it can be observed that the model generates output over a time horizon, sampled at equal time intervals. Therefore, model output is a typical time-series data.

Dynamic behavior analysis can be done in two forms. Firstly, the model behavior

can be compared with a reference behavior. Secondly, underlying pattern characteristics can be extracted and the model behaviors can be grouped according to them. In all forms pattern characteristics such as trends, slopes, periods, amplitudes, means, variations, minima, maxima and so on are of importance.

The term, reference behavior refers to the time-series data that is available either by historical quantitative and/or qualitative observation or hypothesis. Typical tasks involve comparison with a reference behavior are model calibration, structure validity and behavior pattern validity. Since long-term pattern characteristics are of more relevance in system dynamics methodology, a reference behavior can be in the form of generic dynamic patterns such as an exponential increase of an s-shaped decrease. In indirect structure validity testing, these forms of comparisons are frequently done (Barlas, 1996). The model's ability to reproduce the historical time-series is also emphasized in model validation literature in order to increase credibility of the model (Barlas, 1996; Forrester and Senge, 1980; Sterman, 2000; Martinez-Moyano and Richardson, 2013). The tests related to this purpose are defined as behavior pattern validity test (Barlas, 1996).

The straightforward approach to measuring similarity or dissimilarity of two time series is comparing means, standard deviances, or just the final values of two time series. A more advanced approach might be to consider every pairs of data points and obtain a time-series of differences, which is the basis for point-based metrics such as sum of squared errors (*SSE*), mean squared error (*MSE*), R^2 , and Theil statistics (Theil, 1966). However these approaches have the possibility of yielding incorrect results. Two time series having similar pattern characteristics can be statistically shown to be dissimilar, or vice versa according to these basic overall techniques. These approaches may in some cases be suitable; nevertheless for a long-term policy oriented modeling approach, similarity of two time series should be handled by taking the pattern components into account. There is a need for assessing the model's overall pattern prediction and reproduction ability rather than point-based prediction and forecasting ability. For example, a frequently used statistical is metric mean squared, which is calculated as $MSE = \frac{1}{n} \sum (X_A - X_B)^2$. In Figure 3.1, the weakness of

arbitrarily using MSE measure for comparison of behavior patterns is illustrated. In this figure, an arbitrary negative exponential curve outperforms a hypothetical model output with respect to their similarity to actual reference data. The small phase and bias shift results in large penalty due to point-based assessment of MSE .

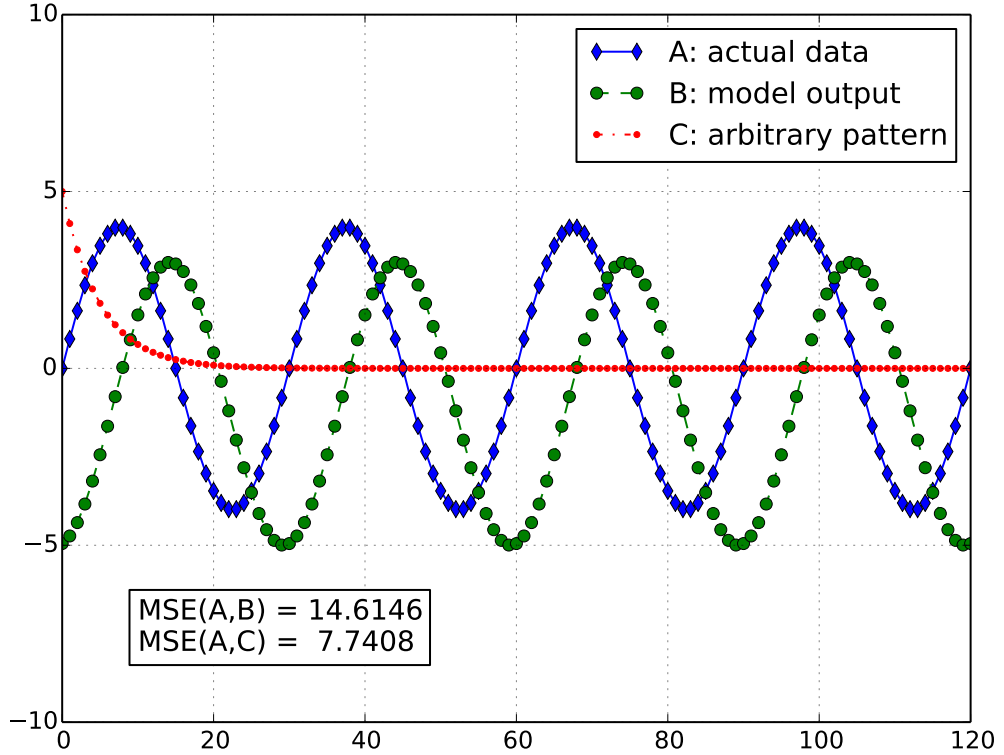


Figure 3.1. Illustrative example for showing weaknesses in point based metrics.

3.2. Literature Review

Behavior analysis and evaluation of model output is in deep connection with time series analysis. The reason for that is that, as stated above, the dynamic system simulation models generate data points in successive time intervals over a predetermined time horizon. The collection of all data points constitute a time-series data. In traditional statistical literature the analysis efforts made on this type of data is referred as *time series analysis*, and also in machine learning and pattern recognition similar efforts are referred as *temporal data mining* or *time series data mining*.

Examples of time-series data can be yearly world population from 1980 to 2052, global average surface temperature recorded monthly from January 1969 to December 2013, inventory levels of an item recorded at every hour during one week.

Time-series analysis had emerged as a field with the efforts of understanding the underlying characteristics of the complex real world data (Yule, 1927). Traditional time series analysis, and it has been extensively applied to economic data, business cycles, weather data, etc. for forecasting and control purposes (Box and Jenkins, 1970). Analysis methods can be differentiated as the ones that work under time-domain e.g. autoregressive models, moving averages, cross-correlations (Box and Jenkins, 1970) and frequency-domain e.g. fourier transforms (Bloomfield, 1976), wavelet methods (Percival and Walden, 2000). In modern time series analysis these two different approaches merged with the use of proper transfer methods (Pollock, 1987; Tsay, 2000). Traditional time-series analysis involves identifying a hypothesized underlying statistical model that would be a good representation of the data, then estimating the parameters of that statistical model, and test goodness of fit with various measures (Bowerman *et al.*, 2005; Peter J. Brockwell, 2006; Mills, 2011). Stationarity of time series is important for most of the analysis methods. A time-series is stationary if its mean and variance are constant over its time horizon. Non-stationarity can be eliminated by trend regression and transfer functions. Also a recent method by Huang *et al.* (1998) decomposes a time series into its so called intrinsic mode functions. This method is proposed for analysis of all types of data; whether stationary or non-stationary, linear or nonlinear (Huang and Atttoh-Okine, 2005).

From the literature review on statistical methods it can be concluded that these methods are not suitable to be used directly for evaluation of dynamic behavior without proper customization and usage procedure. Since they are based on point base computations there is a possibility of coming up with undesired conclusions. Each proposed method can be useful in specific cases. More benefits can be obtained by using them within a logical framework and order. In the system dynamics model validation and analysis literature Barlas (1989, 1996) suggests an iterative analysis procedure which is built on top of available statistical inference techniques. The components of this

procedure are trend regression, autocorrelation functions. They are described in detail in Section 4.3.

In addition to traditional statistical methods, time-series data is the most common and fundamental form of temporal data (Mitsa, 2010). Time series data mining has emerged and prospered with advances in database systems, fast computation technology, availability of more data, and as well as new techniques that enable transforming non time-series like gene sequences into quasi time-series. Time series data mining objectives can be categorized as classification, clustering, pattern motif discovery, pattern anomaly discovery, query search (Mitsa, 2010). In order these objectives to be accomplished there has to be a predetermined metric for similarity assessment. In the literature there are examples of as such euclidean distance, absolute difference, symbolic representation, dynamic time warping (Liao, 2005). Also the algorithms, which in the basic case are developed for classification or clustering on one dimensional data point at a time, have to be adjusted for a time-series data, i.e. sequence of data points. Since, working with raw data is costly, time-series data has to be represented by summarization metrics. They can be categorized into four (Bagnall *et al.*, 2006). According to this categorization data representation can be statistical model based, data adaptive, data non adaptive and data dictated. For a complete review on state of the art methods, research by Fu (2011) can be consulted.

Among the objectives of temporal data mining, two of them, pattern classification and clustering methods, are relevant for the analysis of dynamic behaviors. Classification and clustering methods in pattern recognition are able to provide useful metrics for labeling the different model outputs as discussed in the previous section. Based on the nature of the data and the research objective the best similarity measure, data summarization method to be implemented differ. There are no method that outperforms in all situations. Among large number of approaches, the one that results in best classification or clustering results depends highly on the nature of data and the research objective (Keogh and Kasetty, 2002; Mitsa, 2010).

From the review on pattern recognition literature, it can be concluded that the

combination of data representation technique, data similarity assessment and classification/clustering algorithm must be custom tailored for each application. In this study, the method that is selected to be included in the analysis toolkit is ISTS Algorithm by Kanar (1999). It is a pattern classifier, that uses Hidden Markov Models to measure the class probabilities of time-series which are summarized as a feature vector of mean, slope and curvature.

3.3. Discussion

Literature review can be concluded as there exists two meaningful aspects for model output evaluation.

- (i) There are methods that compare patterns based on statistical properties. In this family of methods there is a possibility of obtaining undesired results for pattern comparison, i.e. similar patterns may be different according to the statistical metric used. These methods should be used in a meaningful logic and framework.
- (ii) There are methods that compare patterns as objects. In this aspect pattern recognition literature provides classification and clustering methods. These principles must be custom tailored for system dynamics model output evaluation.

4. SELECTED METHODS FOR DYNAMIC BEHAVIOR ANALYSIS

In this chapter the selected toolkit for analysis of dynamic behavior is described in detail. The methods are derived from the literature review in the previous chapter. Additions to the literature, if any, are described in its corresponding section.

In Section 4.1 a set of data preparation methods are introduced. In Section 4.2 a pattern classification algorithm, namely *ISTS Algorithm* (Kanar, 1999), for classification of generic dynamic behavior patterns is described. Generic dynamic patterns are defined as 25 theoretically and empirically observed behavior patterns. In Section 4.3 a procedural approach, namely *Multi-Step Procedure* (Barlas, 1996), that is used to compare pattern related characteristics of two time-series is described. This procedure involves several statistical analysis functions and methods that become meaningful for assessing model behavior validity when used in an intelligent order. In addition their usage for behavior validity assessment, the methods can also be used individually for other analysis purposes. Briefly the methods are trend regression, mean, variance, autocorrelation, spectral analysis and cross-correlation. In Section 4.4 improvements for more systematic visual inspection, namely *Enhanced Graphical Inspection* experience is investigated and presented.

4.1. Preprocessing of Dynamic Behavior

Data preparation or preprocessing operations are much broader in the literature; however, in this section relevant and general methods are described. Relevant properties of data in this context are transient and steady-state phases distinction, being stationary or non-stationary, and being noisy or smooth. For a behavior that has all these characteristics -namely a behavior that has both transient and steady-state components, has a trend and is noisy- the correct sequence to prepare data for statistical analysis is: First, separation of transient and steady-state parts; secondly, removal of

trend; finally, smoothing of noise.

4.1.1. Separation of Transient and Steady-State Behavior

In simulation studies the model output is generally analyzed in two phases; transient and steady-state (Shannon, 1975). The transient phase is observed as a result of initial conditions or as a response to disturbances on the system in equilibrium. Typically after certain duration passes, the variables of the model start exhibiting their steady-state behaviors. Depending on the model structure, time horizon, and initial conditions, one of the two phases can be relatively short and negligible.

Also depending on the model and its purpose, the behavior analysis is made either only on one of the phases, or both. Discrete-event simulation studies propose approaches like replication in cases where the effects of transient response are desired to be eliminated (Shannon, 1975; Hoover and Perry, 1989). Replication is useful for analysis where only point-based measures like average queue length are of importance, and the model output changes from simulation to simulation. However in system dynamics modeling, behavior analysis can not be restricted to one point statistics like an average value, rather all the points and their interrelations are important, e.g. periodicities, fluctuations, trends, and also the system dynamics models are deterministic. The model would generate exactly the same dynamic behavior unless the parameters of the model are not changed. So, replication method is inapplicable in this case. In order to conduct accurate behavior analysis on system dynamics models transient and steady-state phases should be separated. For example, linear trend estimation operation would give incorrect results if it is applied to time-series that has transient and steady-state components. For an elegant analysis toolbox, there has to be splitting tools for separation of these phases, therefore enabling analyzing them separately.

4.1.2. Smoothing

Noise reduction issue has a significant place in certain fields like signal processing (Scharf and Demeure, 1991), econometrics (Tsay, 2002) that involve multiple high

frequency nonlinear fluctuations. In order for regression analysis to operate, there are fundamental assumptions on the nature of error terms; independent, zero mean, constant variance, normality. In system dynamics context, the concept of noise is less important. The main reason for this situation is that mathematically system dynamics models are composed of differential or difference equations, which are deterministic in nature (As mentioned above, in certain applications noise can be artificially introduced by the modeler). The most relevant importance of noise is encountered in cases where comparisons of model outputs with real life observations are involved. Generally real data is noisy, since there are unexplainable factors present. In order to be able to analyze pattern related characteristics of such noisy real world data (and also model output if noise is artificially introduced) there has to be practical naive noise smoothers.

In this study two intuitive and frequently used smoothing methods are selected. They are *exponential smoothing* and *m-point moving average*. These methods are also used for forecasting purposes in different disciplines.

In exponential smoothing filtering the next data point is explained by a portion of the previous data point. The formula for exponential smoothing is as follows:

$$\hat{Y}(t) = \alpha \cdot \hat{Y}(t-1) + (1-\alpha) \cdot Y(t), \quad 0 < \alpha < 1$$

In moving average the next data point is explained by the weighted average of n many previous data points. The formula for moving averages is as follows:

$$\hat{Y}(t) = [Y(t) + Y(t-1) + \dots + Y(t-\beta)]/\beta, \quad 0 < \beta$$

4.2. Classification of Dynamic Behavior Patterns Based on ISTS

Algorithm

In this study a pattern classifier is decided on to be included in the analysis toolkit. It is called ISTS Algorithm. It is a Hidden Markov Model (HMM) based pattern classification algorithm. It takes a time series as input, and returns state optimized

likelihood values based on predefined behavior classes. The algorithm is a trained one, namely it can recognize classes that it is introduced before. Currently, given a time series, ISTS Algorithm can recognize six behavior class families, that are observed both theoretically and empirically; constant, growth, decline, growth-and-decline, decline-and-growth, and oscillatory. With variations based on these fundamental patterns, the total number of behavior classes that can be recognized and classified by ISTS Algorithm is twenty-five, all of which can be seen in Figure 4.2. The behavior classes are coded as abbreviations, e.g. *sshgr* represents *s-shaped growth* pattern class. Full list of descriptions of class ids can be found in Table 4.1.

The method was initially developed by Kanar (1999); Barlas and Kanar (1999) for structure-oriented behavior testing (Barlas, 1996). It was used for extreme-condition testing in the form of hypothesis tests stated as such: “*If the model is run under conditions C , then behavior B will result*”. It is shown that ISTS Algorithm performs as good as the expert human eye (Soylu, 2006). It is implemented in other studies for model validation and parameter estimation (Boğ and Barlas, 2005; Yücel and Barlas, 2011).

```

Receive the time-series data,  $\mathbf{X}$ , to be classified
Normalize  $\mathbf{X}$  using min-max normalization procedure to obtain  $\mathbf{X}_N$ 
Split  $\mathbf{X}_N$  into six segments of equal-length, and obtain  $S_1, S_2, \dots, S_6$ 
For each segment  $S_i$  {
    Calculate mean
    Calculate slope
    Calculate curvature
}
For all behavior classes  $B_j$  {
    Calculate the state-optimized likelihood of  $\mathbf{X}_N$  belonging to the class  $B_j$ 
}
Return the results for all behavior classes

```

Figure 4.1. ISTS Algorithm.

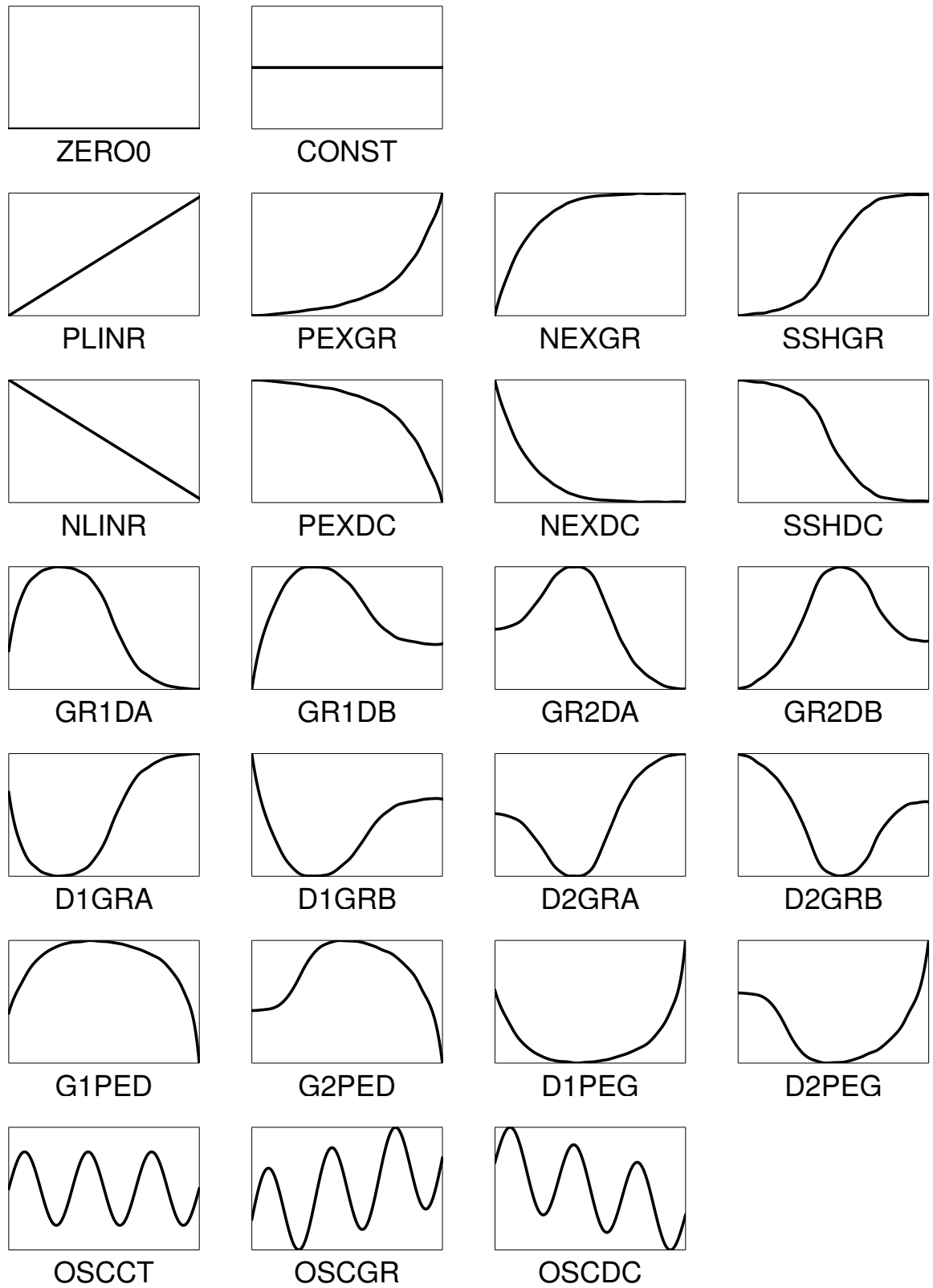


Figure 4.2. Generic dynamic patterns that can be classified by ISTS Algorithm (Kanar, 1999).

Table 4.1. Description of generic dynamic patterns in Figure 4.2 (Kanar, 1999).

Class id	Description
ZERO	Zero
CONST	Constant
PLINR	Linear with positive slope
PEXGR	Positive exponential growth
NEXGR	Negative exponential growth
SSHGR	S-shaped growth
NLINR	Linear with negative slope
NEXDC	Negative exponential decline
SSHDC	S-shaped decline
PEXDC	Positive exponential decline
GR1DA	Growth with decreasing rate followed by decline to equilibrium (growth level is less than decline level)
GR1DB	Growth with decreasing rate followed by decline to equilibrium (growth level is greater than decline level)
GR2DA	S-shaped growth and decline to equilibrium (growth level is less than decline level)
GR2DB	S-shaped growth and decline to equilibrium (growth level is greater than decline level)
D1GRA	Decline with increasing rate followed by growth to equilibrium (decline level is less than growth level)
D1GRB	Decline with increasing rate followed by growth to equilibrium (decline level is greater than growth level)
D2GRA	S-shaped decline and growth to equilibrium (decline level is less than growth level)
D2GRB	S-shaped decline and growth to equilibrium (decline level is greater than growth level)
G1PED	Growth with decreasing rate followed by positive exponential decline
G2PED	S-shaped growth followed by positive exponential decline
D1PEG	Decline with increasing rate followed by positive exponential growth
D2PEG	S-shaped decline followed by positive exponential growth
OSCCT	Oscillation around constant mean
OSCGR	Oscillation around linearly growing trend
OSCDC	Oscillation around linearly declining trend

In order to illustrate the input and output relations of the ISTS Algorithm four example dynamic behavior patterns are illustrated in Figure 4.3. In this figure, a linear growth, an s-shaped growth, a growth then decline, a goal seeking decay and an oscillatory patterns are fed to the algorithm respectively. The algorithm returns twenty-five likelihood values for each input pattern, which can be observed in Table 4.2. In this table of likelihoods, the small values indicate that the input behavior pattern is less likely to belong to that behavior class, whereas the largest values (depicted as bold in the table) indicate the most likely behavior class for the input time-series pattern. If the input behavior is not similar to any of the predefined behavior classes then all of the likelihood values would result in larger negative numbers.

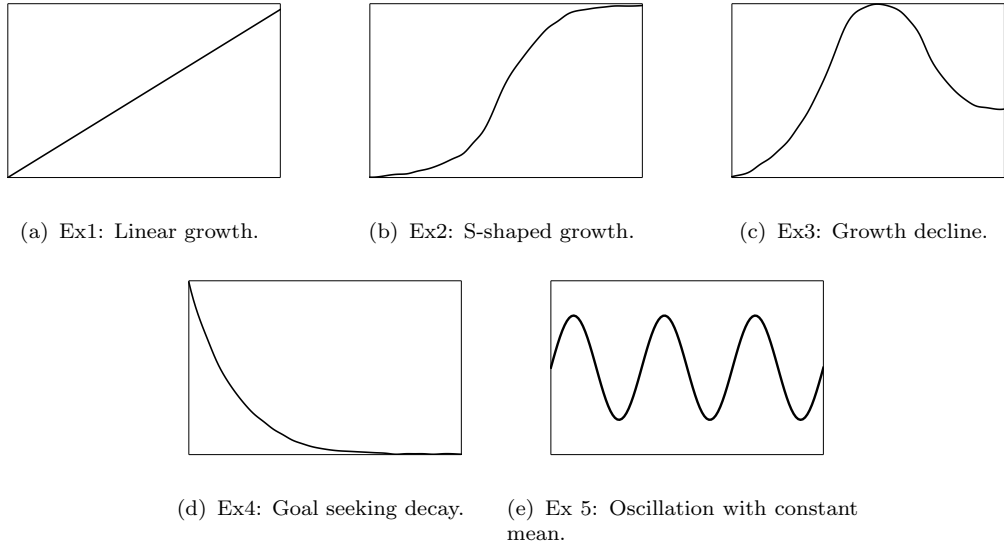


Figure 4.3. Dynamic behavior pattern examples.

In ISTS Algorithm, classes of the constant family (*zero0* and *const*) and oscillatory family (*oscct*, *oscgr*, *oscdc*) are treated differently than the rest of the classes. For evaluation of likelihoods regarding these two families, the algorithm makes a binary yes-no comparison. As a result, likelihood value of *oscct* class for non-oscillatory behaviors return as equal (see first four examples in Table 4.2). For full details on ISTS Algorithm see Kanar (1999) and Soylu (2006).

Table 4.2. ISTS Algorithm results for behaviors in Figure 4.3.

Class	Ex1	Ex2	Ex3	Ex4	Ex5
ZERO0	-10.0000	-10.0000	-10.0000	-10.0000	-10.0000
CONST	-10.0000	-10.0000	-10.0000	-10.0000	-10.0000
PLINR	-0.1454	-6.2835	-10.5077	-12.7413	-12.7400
NLINR	-12.7413	-12.7413	-11.0656	-9.0685	-12.7400
NEXGR	-5.2919	-5.6883	-8.5316	-15.5448	-14.3400
SSHGR	-3.4732	1.2116	-11.3223	-19.2003	-21.4400
PEXGR	-5.4142	-3.5477	-12.8769	-10.6131	-18.2100
GR1DA	-8.5844	-9.7281	-10.4846	-2.2153	-9.1840
GR1DB	-11.8988	-12.9673	-6.3872	-6.6042	-15.4200
GR2DA	-6.1521	-6.4692	-2.8863	-2.1856	-9.1090
GR2DB	-1.8293	-2.4017	-0.1145	-7.1643	-12.6400
D1PEG	-8.8071	-6.5458	-10.8852	-4.0611	-13.2300
D2PEG	-8.4865	-8.5306	-11.1068	-1.8091	-14.5700
NEXDC	-10.7787	-11.5605	-11.2122	-0.1175	-14.0200
SSHDC	-16.7834	-21.0931	-19.8869	-2.4655	-18.6000
PEXDC	-14.6493	-15.4032	-13.3925	-14.8529	-18.0500
D1GRA	-4.4429	-1.3771	-9.4737	-6.4075	-10.5900
D1GRB	-7.1409	-2.8407	-10.1601	-2.5886	-14.1600
D2GRA	-4.2208	-1.5577	-8.9187	-4.1252	-9.3150
D2GRB	-4.0185	-6.0475	-5.0064	-4.6070	-11.9100
G1PED	-11.5694	-10.2605	-8.4474	-13.3114	-12.9600
G2PED	-6.7173	-7.5241	-5.7460	-15.1670	-14.2300
OSCCT	-7.2421	-7.2421	-7.2421	-7.2421	-1.7040
OSCGR	-7.2421	-7.2421	-7.2421	-10.0000	-10.0000
OSCDC	-10.0000	-10.0000	-10.0000	-7.2421	-10.0000

4.3. Multi-Step Procedure of Behavior Testing Software (BTS)

The functions and methods presented in this section are based on the works of Barlas (1989, 1990, 1996); Barlas and Erdem (1994); Barlas *et al.* (1997); Bozyayla (2001), which are related to behavior pattern testing of system dynamics model behaviors. The methods presented in this section are specifically relevant for comparison of steady-state and periodic behaviors. Originally they have been developed for behavior validity testing of system dynamics models. In this thesis study they are proposed for other model analysis purposes in addition to behavior validity, since each individual method enables extracting information on pattern components of a dynamic behavior. These pattern components such as periods and amplitudes can also be used as policy measures and/or calibration objective.

Table 4.3. Multi-Step procedure by Barlas.

MULTI-STEP PROCEDURE	
1	Trend comparison and removal
2	Period comparison using autocorrelation and spectral density function
3	Autocorrelation test for lag comparison
4	Comparing the averages
5	Comparing the variations
6	Amplitude comparison
7	Testing phase lags using crosscorrelation function
8	Overall summary measures: coefficient of similarity

4.3.1. Trend Regression and Removal

In statistical inference it is important to remove trends for fulfilling the stationarity assumption. Searching for three lower order of trends linear, quadratic, exponential is considered to be enough for the purpose of this study, since removal of higher order trends (higher order polynomials, spline curves) may result in loss of significant pattern

related information. If a significant trend is found, it should be removed in order to obtain the de-trended and a possible stationary version of the output.

Trend is also one of the fundamental elements of a behavior pattern. Trend information, i.e. slope, intercept, gives quantitative measures for comparison of two behavior patterns. The questions like “Are the trends of two time series are of similar shape, linear or exponential?” can be answered in this step.

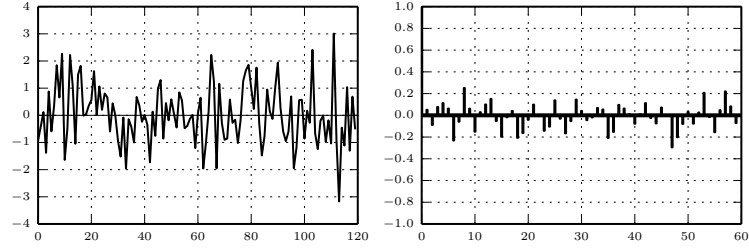
The following table shows the trend types, trend-fitting and trend-removal formulas, where t denotes time index, y input time-series, $\hat{y}$ fitted curve, and y' de-trended curve:

Table 4.4. Trend fitting and removal formulas

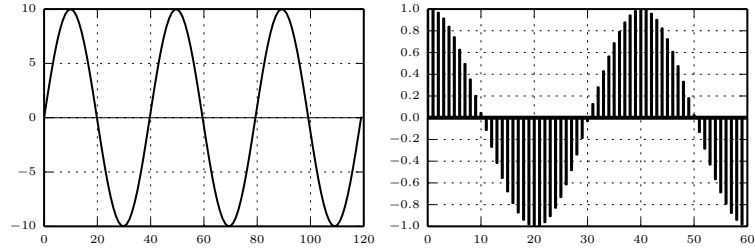
Trend	Fitting Formula	Removal Formula
Linear	$\hat{y} = a.t + b$	$y' = y - a.t$
Quadratic	$\hat{y} = a.t^2 + b.t + c$	$y' = y - a.t^2 - b.t$
Exponential	$\hat{y} = a. \exp^{b.t}$	$y' = y - a.(\exp^{b.t} - 1)$

4.3.2. Autocorrelation

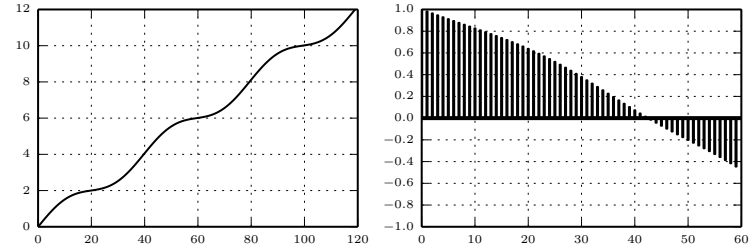
Autocorrelation function gives a measure of how successive data points of a time-series are interrelated (Box and Jenkins, 1970). The function is always 1 at lag zero, and takes values between -1 and 1 at other lags since it is a *normalized* function. Autocorrelation function immediately drops to zero for uncorrelated (random) series. If the series are moderately correlated it reaches zero in a negative exponential fashion. If there exists a trend in the mean (non-stationarity) it reaches zero very slowly. If the time series is cyclic, autocorrelation function is also cyclic (Box and Jenkins, 1970). In Figure 4.4 a random data, a sine wave and a sine wave with linear trend imposed are given. In the oscillatory case, autocorrelation function works as a smoothing operation. The last data indicates that, trend removal operation is necessary for correct interpretation of autocorrelation function.



(a) Random data.



(b) Oscillation around constant mean.



(c) Oscillation with linear trend.

Figure 4.4. Autocorrelation functions for random, oscillatory and non-stationary data with trend.

$$Cov(k) = \frac{1}{N} \sum_{i=1}^{N-k} (x_i - \bar{x}) \cdot (x_{i+k} - \bar{x}) \quad k = 0, \dots, N-1$$

$$r(k) = \frac{Cov(k)}{Cov(0)} = \frac{Cov(k)}{Var(x_i)}$$

There are two usage modes of autocorrelation function. Firstly, for cyclic series, period information can be extracted automatically from the autocorrelation function. Selected period finding algorithm is provided in Figure 4.5.

Secondly, two time series can be compared by statistical hypothesis testing of the differences of the two autocorrelation functions. This method was suggested by Barlas

```

Get the autocorrelation function values  $r(k)$ 
Get the difference vector  $d_i = r_{i+1} - r_i$ 
Get signum function of the difference vector  $s_i = \text{sgn}(d_i)$ 
Set  $i = 1$ 
Repeat until the peak is found or the end of array  $r(k)$  is reached {
    if  $s_i \leq 0$  then peak is found
    else  $i = i + 1$ 
}
Return  $period = i$ 

```

Figure 4.5. Period finding algorithm based on autocorrelation function.

(1990). Let the autocorrelations of two time series are calculated as $r_A(k)$ and $r_S(k)$. The difference vector of these two series is simply $d(i) = r_A(k) - r_S(k)$, for $i = 0, \dots, k$. For testing the null hypothesis $H_0 : d(i) = 0, \forall i$ subject to $H_1 : \text{at least one } d(i) \neq 0, \forall i$. In order to construct a confidence interval an approximation formula for standard deviation of the differences is needed. For this purpose Anderson's (1982) formula is used (Barlas, 1990).

$$Var(r(k)) = \frac{1}{N \cdot (N + 2)} \sum_{i=1}^{N-1} (N - i) [r(k - i) + r(k + i) - 2r(k)r(i)]$$

4.3.3. Spectral Density Function

Spectral densities are representation of data in the frequency domain (Granger and Hatanaka, 1964). The time series is represented as the integration of sine and cosine waves. In the discrete case integration become summations.

The modification to original algorithm by Bozyayla (2001) enables searching in equal period intervals rather than the original search which is on equal frequencies. The motivation for this decision is that in system dynamics more interest is put on the time domain rather than frequency domain (See Bozyayla (2001) for a detailed discussion).

The method not only reveals the main period but also provides useful information on secondary and higher order periods. In Figure 4.6, there is a time-series data that

involves two periodicities is illustrated. The two periodicities are formed of two sine waves with $period = 40$ and $amplitude = 6$, and $period = 10$ and $amplitude = 3$. It can be seen that spectral density function successfully finds the two periods in that is embedded in the input data.

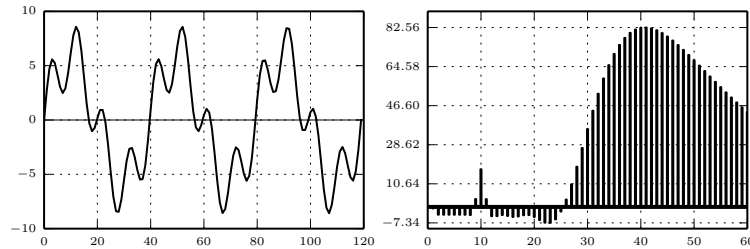


Figure 4.6. Spectral densities of a 2-period oscillation.

4.3.4. Amplitude Estimation

Amplitudes of two oscillatory behaviors can be compared for similarity assessment, or the amplitude of a specific model output can be extracted for policy analysis purposes, i.e. “What is the optimum management policy, that minimizes the amplitude of inventory cycles?”.

For cases where amplitudes of cycles are constant over time, trigonometric function fitting can be used (Koopmans, 1974). Trigonometric regression takes a time-series and guessed or estimated period as inputs and fits a sine curve to the time-series. From the fitted curve, the amplitude, phase and bias values can be estimated. The advantage of this procedure is that it is fast and straightforward to apply. The shortcomings of this method is that the period must be estimated or guessed beforehand, since it is used as an input for the calculations of the method. Also in some cases, the sine wave may not be an adequate representation of the periodic function (i.e. periodicity does not strictly imply oscillation). In these cases, amplitude estimation by modified Multiplicative Winter’s Forecasting method is proposed in the literature (Barlas *et al.*, 1997).

Trend in amplitude is related with trend in variations of the data, therefore

trend regression methods that are applied to overall data can not detect trend in amplitudes. This is practically because that the lower order trend regression functions estimate the overall trend, i.e. trend in the mean. For an oscillatory behavior the trend in amplitude can be constant (stable oscillations), increasing (growing oscillations), or decreasing (damping oscillations), independent of the trend in mean. Moreover increasing/decreasing rate of the amplitudes can be of linear or nonlinear nature.

There are two main approaches for estimation of trend in amplitude both of them requiring that the periods of the cycles are constant over the time horizon and that constant period is calculated/estimated before using the trend in amplitude estimation methods.

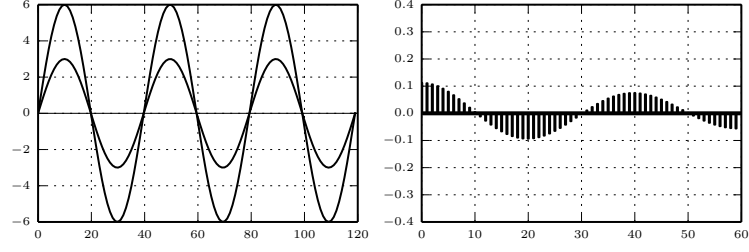
As a common step the time-series are divided into segments according to the period, and therefore data points for each cycle are separated. First approach is to form subsets by grouping data points that are in the same cycle. The second approach is to form subsets by taking every n th data point from each cycle. See Bozyayla (2001) for details of these methods on estimation of trend in amplitude.

4.3.5. Cross Correlation

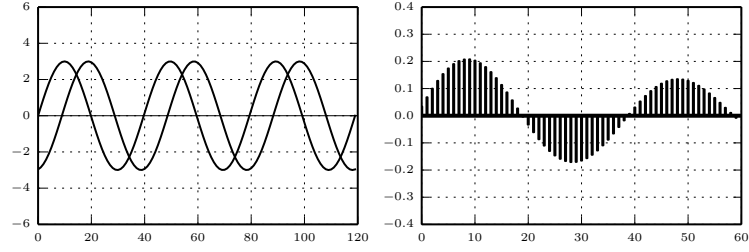
Cross-correlation measures the similarity of two time series as a function of time-shift between them. It be used for comparing the phase lag of two series. Occurrence of max lag at 0 indicates a phase match, and at any other value indicate that the two series are out of phase. In Figure 4.7 in phase and out of phase examples are provided.

$$C_{SA}(k) = \frac{\frac{1}{N} \sum_{i=k+1}^N (S_i - \bar{S})(A_{i-k} - \bar{A})}{s_S a_A}$$

$$\text{where } s_S = \sqrt{\frac{1}{N} \sum_{i=1}^N (S_i - \bar{S})^2} \text{ and } s_A = \sqrt{\frac{1}{N} \sum_{i=1}^N (A_i - \bar{A})^2}$$



(a) Two signals in phase.



(b) Two signals out of phase.

Figure 4.7. Cross correlation plots for two pairs of signals.

4.3.6. Coefficients of Similarity

Having discussed that traditional single value statistics do not provide healthy measures for output evaluation of dynamic simulation models, it is still desired to come up with single value summary statistics. It is easier and faster to communicate the results with them. For this purpose a sum of squares type calculation can be used for comparison of two time series. It should be noted that this value provides meaningful results only if the previous behavior pattern characteristics mentioned above indicate similarity between two time series. Discrepancy coefficient (Barlas, 1989) function takes values between 0 and 1, where 0 indicates the highest similarity. Also see Sterman (2000, p. 875-876) for a nice summary on single value summary statistics.

$$U = \frac{\sqrt{\sum_{i=1}^N ((S_i - \bar{S}) - (A_i - \bar{A}))^2}}{\sqrt{\sum_{i=1}^N (A_i - \bar{A})^2} + \sqrt{\sum_{i=1}^N (S_i - \bar{S})^2}}$$

$$MSE = \frac{1}{N} \sum_{i=1}^N (S_i - A_i)^2$$

$$U^M = \frac{(\bar{S} - \bar{A})^2}{MSE}; \quad U^S = \frac{(s_S - s_A)^2}{MSE}; \quad U^C = \frac{2 \cdot (1 - r) \cdot s_S \cdot s_A}{MSE}$$

$$\text{where } r = \frac{1}{N} \sum \frac{(S_i - \bar{S})(A_i - \bar{A})}{s_S s_A}, \text{ and } s = \sqrt{\frac{1}{n} \sum (X_i - \bar{X})^2}$$

4.4. Graphical Inspection

Comparison of two or more data arrays cannot be completed without proper manual and visual approaches. For transient model outputs, which are in fact encountered on significant portion of system dynamics projects, no general statistical tests are applicable (Barlas, 1996). A popular example for transient dynamics is the “diffusion curves” (Bass, 1969; Mahajan *et al.*, 1990). In such cases, problem specific quantitative measurements are developed and utilized. This kind of analysis tends to be qualitative. Here it is relevant to discuss the notion of key points. For example, for a generic case of the aforementioned diffusion curve, one can define the key points as inflection point, minimum level and maximum level. Nevertheless, for a different type of transient behavior, e.g. growth then decline, the key points would be different. In addition to that, even if the fundamental dynamics are of the same type, depending on the model purpose different measures might be of importance.

Generically, key characteristics of interest can be grouped as maxima, minima, distances, and inflection points as mentioned by Barlas (1996). In this thesis study, we have searched for even more generic representations, and come up with the following categorization.

Generic key characteristics can be of the following types in space-time:

- (i) X-axis points comparison, typically for comparing exact points in time

- (ii) Y-axis points comparison, typically for comparing exact levels of the variable
- (iii) X-axis distances comparison, typically for comparing time durations
- (iv) Y-axis distances comparison, typically for comparing distances or gaps the variables exhibit

The method presented here is an implementation-related one. Therefore detailed description is given in the next chapter, in Section 5.5.7.

4.5. Summary of Behavior Analysis Methods and Procedures

A quick summary for the three groups of analysis techniques can be stated as such: The first method, *ISTS Algorithm* works for all types behavior modes and provides a high level quantitative information on the input related behavior pattern. The algorithm returns the likelihood of an input behavior belonging to a predefined behavior class. For example, it returns by concluding the behavior belongs to s-shaped growth. However the algorithm does not indicate any detailed information on specific properties of s-shaped growth characteristics such as the inflection point and time, maximum level, and saturation time. For more detailed analysis, the third tool *Enhanced Graphical Inspection* is proposed to the modeler. For another case let us consider a situation where the first tool returns results indicating that the behavior belongs is oscillatory. The results do not give information on whether the behavior exhibits a borderline stable oscillation (i.e. constant amplitude over time) or damping oscillation (i.e. decaying amplitude over time) or unstable oscillation (i.e. growing amplitude over time). Consider the case where the decaying amplitude case, there are questions like are the amplitudes decreasing in a linear fashion or an exponential fashion? Similarly the algorithm can not provide information on whether the cycles involve secondary periodicities or not. For all of these cases the second tool *Multi-Step Procedure* can be used to calculate the various pattern information. The elements of the second procedure is specifically designed for oscillatory type of behaviors. They offer rich and automated analysis options.

ISTS Algorithm provides high level quantitative information for generic behavior

types and is eligible for task-level automated analysis.

Multi-Step Procedure provides detailed level quantitative information on steady-state behaviors and is eligible for functional-level automated analysis.

Graphical Inspection provides detailed level quantitative information for all behaviors, especially for transient behaviors, and is eligible for manual analysis.

5. DEVELOPMENT AND PROPERTIES OF BATS

In this chapter the development process of the final software is described and last three items of research procedure is covered. These are namely software implementation of behavior analysis toolkit, integration to existing modeling software and efforts for improving the user experience are presented in this chapter. The software is named as BATS (Behavior Analysis and Testing Software). First, information on the programming language and the selected modeling software, and main design elements of the software are given. Then, there are three sections devoted to detailed description of data inputting and preparation features, data analysis features, and model analysis features. Data analysis features are the implementations of the functions and methods presented in the previous chapter. The usage examples of BATS are covered in the next chapter.

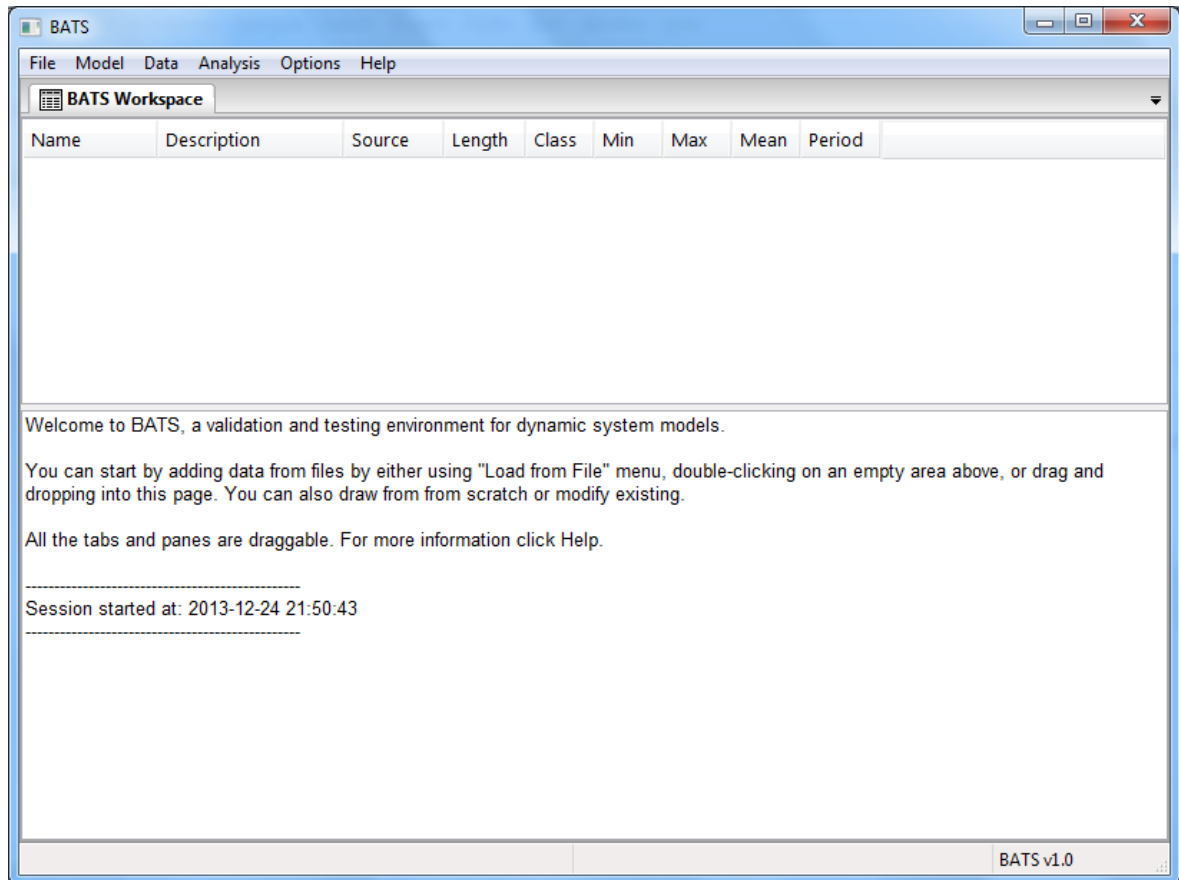


Figure 5.1. Main screen of BATS.

5.1. Programming Language Selection

BATS is written on pure Python³, a high level programming/scripting language. It is an object-oriented dynamically typed language. Noting that the advantages and disadvantages of selecting a specific programming language depends on the purpose of the project, the reasons for choosing Python as the base language of this study are provided in the following lists.

Python vs. compiled languages such as C, C++, Java, Fortran (Prechelt, 2003; Oliphant, 2007):

- Python is much easier to learn and master, since it is a dynamically typed scripting language.
- Python encourages and enables clean, neat and readable coding, which is important for future improvements.
- Python is open source, and has a committed user/developer base.

Python vs. other data analysis environments such as Matlab, Mathematica, R (Pérez *et al.*, 2007):

- Python is free and open source, that makes it flexible and customizable.
- It is impossible to make fully functional graphical user interfaces with the others.
- Since Python is a complete programming language of its own, there are no functionalities that others perform and Python can not; however, certain operations are impossible to do with the others. The most relevant example regarding this study is the allowance of communication with modeling software.
- Combination of three open source and widely used libraries covers all the essential capabilities of Matlab environment; Numpy and Scipy are used for computational and algorithmic aspects and Matplotlib is used for plotting aspects.
- Software can be developed, licensed and distributed freely using Python.

³Available at <http://www.python.org>

The only disadvantage presented for Python, is its computational performance compared to compiled languages. This main disadvantage can be compensated by implementing a hybrid coding strategy. It is possible to transfer computational bottlenecks into one of the compiled languages C, C++ or Java (Langtangen, 2009). Python can also be considered as a practical *glue* language that binds code parts from other languages.

5.2. Communication with Modeling Software

In this study, Vensim is selected as the first modeling software to be integrated to the developed software. Main reasons for selecting this particular program are:

- Vensim is one of the three core software⁴ that is used to build and simulate system dynamics models. The other two are Stella/iThink and Powersim Studio. It is reported that the majority of published work in the field has been done using these three tools.
- Vensim provides the necessary tools for custom-designed software applications (via Vensim dll) that can communicate with models.
- Vensim is free for academic uses.

Crucial operations, namely getting model variables, sending commands for setting model parameters running the model, and finally getting the results of variables of interest can be successfully done with this connection. As a side note, it is proper to state that currently only changing the values of constant parameters are allowed, i.e. editing the structure of the model and modifying the shapes of the look-up functions are not allowed. They had to be done in the original software.

⁴<http://tools.systemdynamics.org/core-sd-software>

5.3. Main Design Elements

Figure 5.2 illustrates main building blocks of BATS and flow of fundamental programming objects. In BATS, the user's interaction with the system happens via *Graphical User Interface (GUI)*. *GUI* displays all kinds of visual information that are stored in objects called *Tabs*. *Tabs* are individual pages of visual information that can co-exist with others. They are stored and managed in *Plotbook*. The *Tabs* are created by two families of analysis operations one family for data related features and one for model analysis features. Data related features work with *DataItem*, an object that keeps time-series data as well as other statistical and descriptive information. *DataItems* can be imported from external data files or created within BATS by both the data related features and model analysis features. They are stored and managed in *Workspace*. Model analysis features communicate with the Vensim software and create specialized *Tabs*. The user controls the program via *GUI* and the commands and events are distributed to corresponding blocks. In the following pages more detailed information for each building block and object are given.

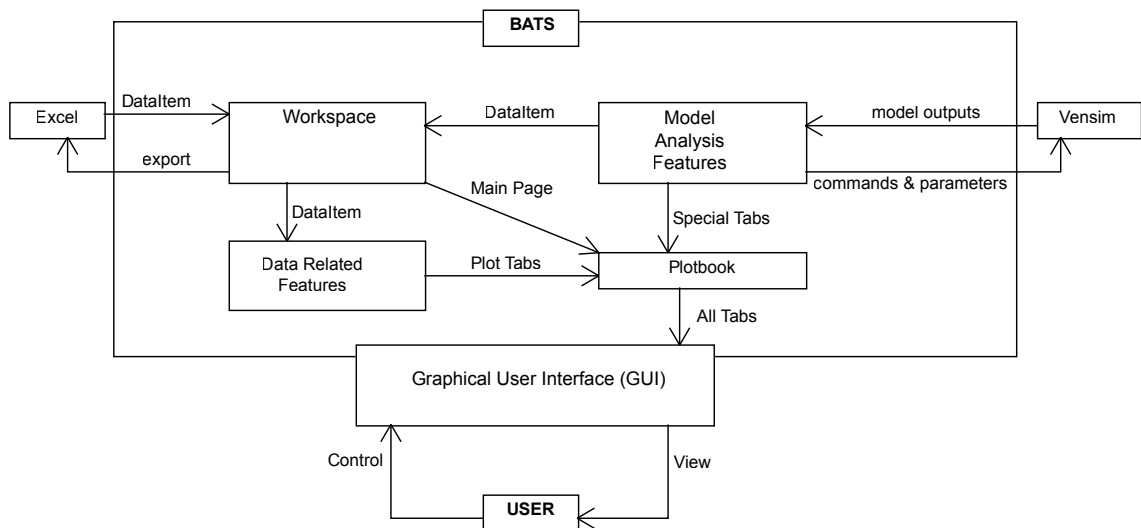


Figure 5.2. Overview of Elements of BATS.

5.3.1. Graphical User Interface

For model analysis environment designed in this thesis study, it is desired to enable the modeler to work with multiple screen views. This is suggested as an element to enhance usability and effectiveness of the overall analysis session. In order to implement this principle in this study, firstly available interface design traits are analyzed and then the appropriate one is selected.

User interfaces can be basically and historically grouped into two categories; single document interface design (SDI) and multiple document interface design (MDI). In SDI, only one window is connected to one instance of the program. While it is applicable to single document programs such as word processors and presentation preparation software, it is not suitable for this study since it does not allow multiple screen views. Also, an example application of SDI can be seen in Figure 5.3(b), a screen view from SiS (Boğ and Barlas, 2005). In this software multi tasking is not enabled at all, and the application needs to be restarted for each major operation, in this case performing a simulation experiment.

Historically, enabling multiple screen views objective have been reached by implementing MDI design in applications. In MDI, multiple floating windows are allowed to be created and controlled by the same one instance of the application. It has been observed that there is a major disadvantage related with these floating windows. This design degraded the user experience due to the difficulties experienced in managing the windows. Among the problems, the windows' easily getting lost and piling up of windows can be stated. MDI design is still implemented in some regular programs like MS Excel, but in a more modified and hidden way. The problematic issue of piling up cascading objects in MDI design can be seen in Figure 5.3(a), a screen view from BTS (Barlas *et al.*, 1997). In MDI, the increasing number of plots tend to pile up and degrade usability.

Modern, state of the art software applications which necessitates working with multiple screen views are observed to implement tabbed document interface design

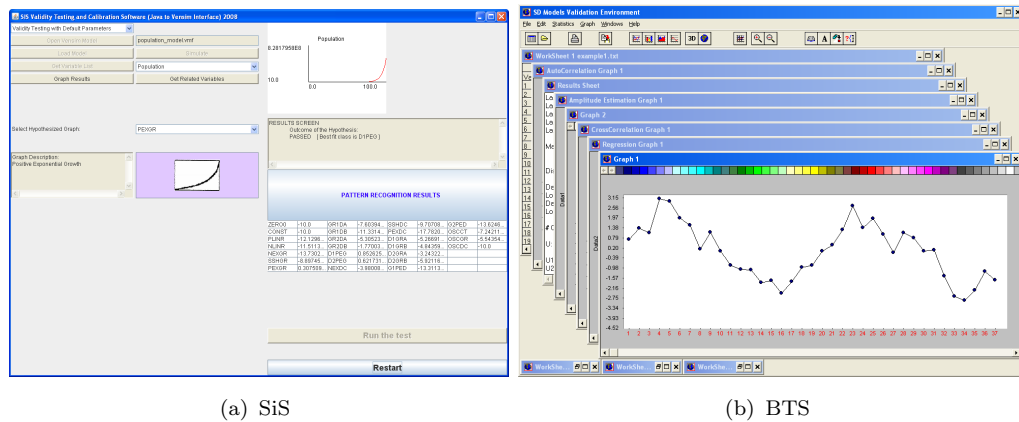


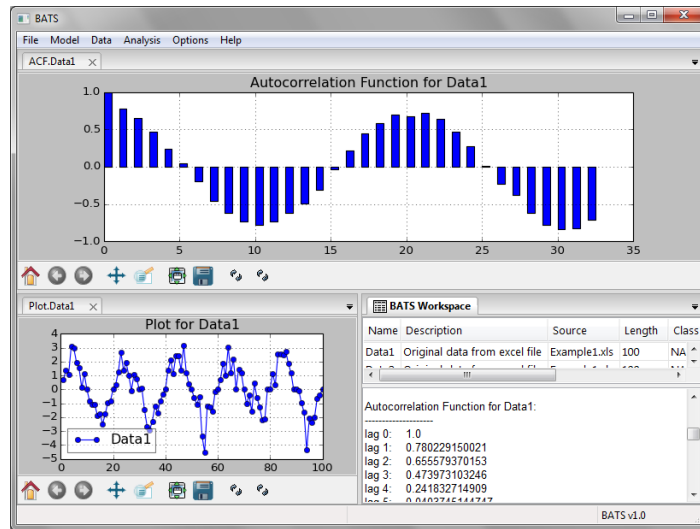
Figure 5.3. SDI design of SiS (Boğ and Barlas, 2005) and MDI design of BTS (Barlas *et al.*, 1997).

(TDI). TDI was popularized in 2000s by web browsers like Firefox and Chrome. In TDI, multiple screens are arranged in the form of tabs as opposed to free floating windows in MDI. This implementation is observed to enhance control and usability of the applications. In addition to basic TDI, text and graphics editors such as Eclipse, Visual Basic, Photoshop, Illustrator, as well as Matlab make use of customized enhanced versions of TDI which as a category is referred as IDE-style (Integrated Development Environment style) Interface. In this design some form of TDI is displayed together with other panes, panels, toolbars, messages, logs, interactive shells, and so on depending on the purpose of the application.

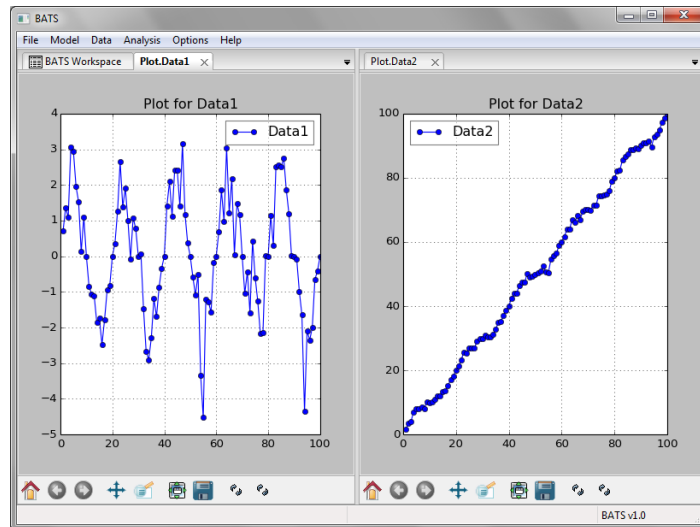
BATS implements an enhanced version of TDI design utilizing wxPython⁵ 's wxAUI⁶ framework, which enables the following capabilities; splittable and dockable floating frames, customizable floating/docking behavior, customizable look-and-feel. It is technically possible to change organization of *Tabs* and docking/floating them while using BATS. This enhances overall usability of the software by introducing customization and flexibility. Figure 5.4 illustrates three representative custom views that can be achieved with the capabilities of wxAUI framework.

⁵<http://www.wxpython.org>

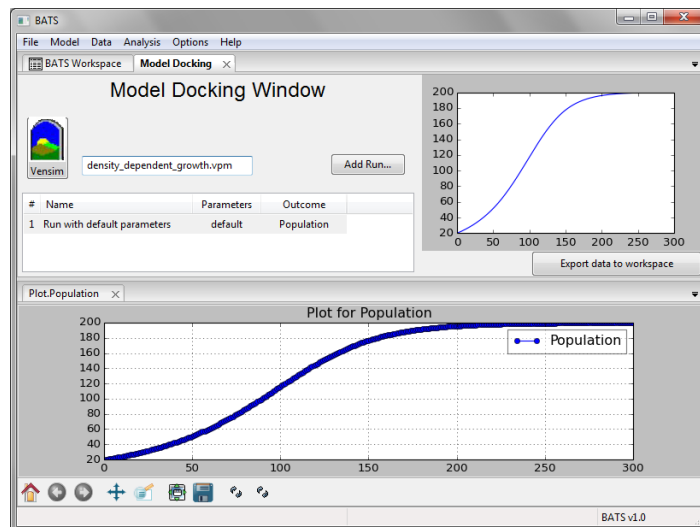
⁶<http://www.kirix.com/labs/wxaui.html>



(a)



(b)



(c)

Figure 5.4. Representative customized views of BATS.

5.3.2. Workspace and DataItems

Workspace is the data management and storing system in BATS. It provides the inputs for data analysis functionalities. It consists of items, each of which are from class *DataItem*. User can add items to the *Workspace* using the following methods:

- Importing from a text file (.txt)
- Importing from an Excel spreadsheet (.xls, .xlsx)
- Reading from a Vensim output file, provided that the name of the parameter is known (.vdf)
- Running a Vensim model by setting parameters, and exporting results (.vpm)
- Drawing a data series manually using the plotting pod in the software
- Extracting sections of data from an existing item in the workspace using *Split* or *Select* tools
- Performing one of the data preprocessing methods; de-trending and smoothing

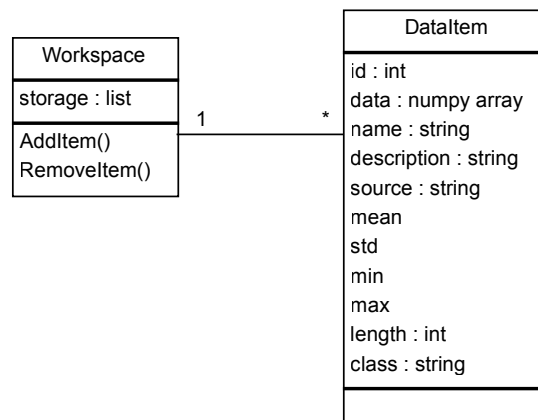
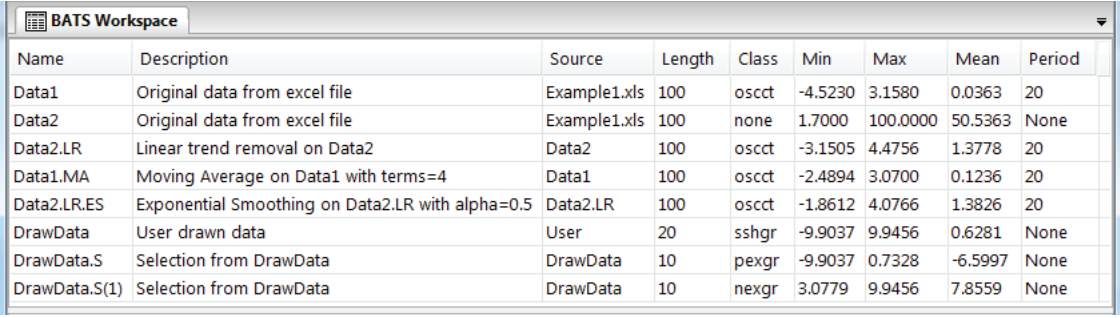


Figure 5.5. The class diagram for *Workspace* and *DataItem*.

Workspace is connected to the *GUI* via a list viewer. In this list, the name of the item as well as other information are shown to the user. The information consists of descriptive information (name, source, description, length), statistical information (mean, var, std, max, min, median), and pattern related information (behavior class and fundamental period). These fields are filled automatically and aim to provide quick

insights to the user. Name and description fields are editable for customization. The pattern related information is calculated automatically and if such information does not exists or can not be found then *none* denoting “information not available” is displayed. The program follows an intuitive naming convention as such: Linear trend removal operation on *Population*, results in *Population.LR* ; exponential smoothing operation on *Population.LR*, results in *Population.LR.ES*, etc. Also in order to maintain a unique name for each item, a renaming algorithm is implemented. Figure 5.6 illustrates the information given here. It should be read from top to bottom, following a chronological order of a an example analysis session. Finally, the user can export the *DataItems* created during the analysis session to an Excel file using *Save Workspace* command from the menu.



Name	Description	Source	Length	Class	Min	Max	Mean	Period
Data1	Original data from excel file	Example1.xls	100	oscct	-4.5230	3.1580	0.0363	20
Data2	Original data from excel file	Example1.xls	100	none	1.7000	100.0000	50.5363	None
Data2.LR	Linear trend removal on Data2	Data2	100	oscct	-3.1505	4.4756	1.3778	20
Data1.MA	Moving Average on Data1 with terms=4	Data1	100	oscct	-2.4894	3.0700	0.1236	20
Data2.LR.ES	Exponential Smoothing on Data2.LR with alpha=0.5	Data2.LR	100	oscct	-1.8612	4.0766	1.3826	20
DrawData	User drawn data	User	20	sshgr	-9.9037	9.9456	0.6281	None
DrawData.S	Selection from DrawData	DrawData	10	pexgr	-9.9037	0.7328	-6.5997	None
DrawData.S(1)	Selection from DrawData	DrawData	10	nexgr	3.0779	9.9456	7.8559	None

Figure 5.6. The appearance of *Workspace* after a hypothetical analysis session.

In addition to storing and managing *DataItems*, *Workspace* also manages the log screen of BATS, which is called as *Logbook*. It is the area where detailed information as a result of function calls are reported to the user and stored throughout the analysis session. The logbook is updated after each operation.

5.3.3. Plotbook and Tabs

Tabs are the fundamental objects for visual display in BATS. They are created after a functionality is called by the user. Typically a tab consists of the following pieces for *Data Analysis Features* as well as *Data Inputting and Preparation Features*; (i) plots with axes, lines, title, legend, plotting tools and options, (ii) buttons, (iii) sliders, and (iv) other function specific controls . For *Model Analysis Features* the *Tabs* are highly

specialized for the specific purpose of the feature.

The storage and management of tabs are handled by *Plotbook*. The first tab of the *Plotbook* is referred as the *Main Page*. It is a special tab in which there are no graphical plots, and two of the main design elements the Workspace and the Logbook are displayed and updated with various operations. This tab is a permanent tab in the *Plotbook*, it is created at the initialization step of the program and stays throughout the entire session. The rest of the *Tabs* include various screens from simple plots as *Plot* feature to complex structures such as *Model Docking Window*. They are created and deleted on demand. In Figure 5.7 class diagram for *Plotbook* and *Tab* can be seen.

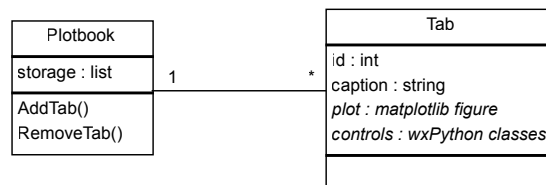


Figure 5.7. The class diagram for *Plotbook* and *Tab*.

5.4. Data Inputting and Preparation Features

In this section data inputting and preparation features of BATS are described. This section is specifically related to creation of *DataItems* both from external or internal operations. The screenshot of *Data* menu is shown in Figure 5.8. In this section detailed information for each menu item is presented.

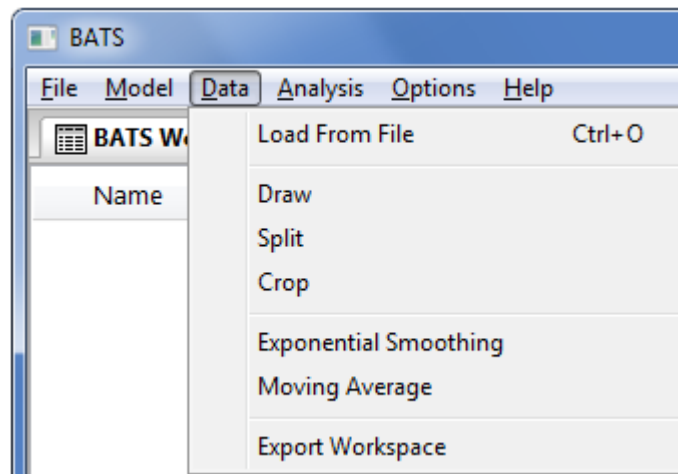


Figure 5.8. *Data* menu for data inputting and preparation features.

5.4.1. Load Data From File

This feature is generally the initial step before using other rich features of BATS. It can be called by three methods; (i) from *Load from File* menu, (ii) drag and dropping the data file into the interface, and (iii) double-clicking on an empty space on the main page. The program automatically extracts the file extension and determines the file type. BATS currently is compatible with three types of files. These are spreadsheet files (.xls, .xlsx), text files(.txt), Vensim results files (.vdf). Excel and text files are assumed to be populated on a data-per-column basis, i.e. one column corresponds to one time-series data. The headers and data point are internally located upon function call. The headers are shown to the user so that the user manually selects specific data series or imports all of the available data using *Import All* command. For Vensim results files the data can be imported if the exact name of the model variable is known. Communication with Vensim models are conducted in a more flexible manner via the

Model Docking Window feature, which is described in Section 5.6.1.

5.4.2. Draw

Draw feature is used for drawing a time-series data on the coordinate system manually. In cases where it is impossible to find detailed historical or empirical data, but there exists qualitative information, this feature proves useful.

Draw feature is initiated with the custom dialog box shown in Figure 5.9. Via this dialog box, user specifies minimum and maximum points for x and y coordinates. Also the number of data points is taken as input from the user, which is then used for dividing the x axis into equal slices. The user draws the data points by manually clicking on the plot canvas. In order to maintain there is exactly one y-value for an x-value there is an internal algorithm, which checks the points as they are added. If a new y-value is added to an existing x-value then the previous value is updated with the new value. Finally, the user finishes the drawing by clicking on the *Save* button. This operation results in creation of a new *DataItem* to be stored in *Workspace*.

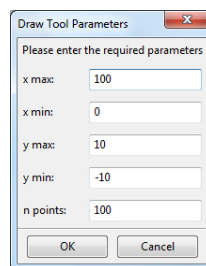
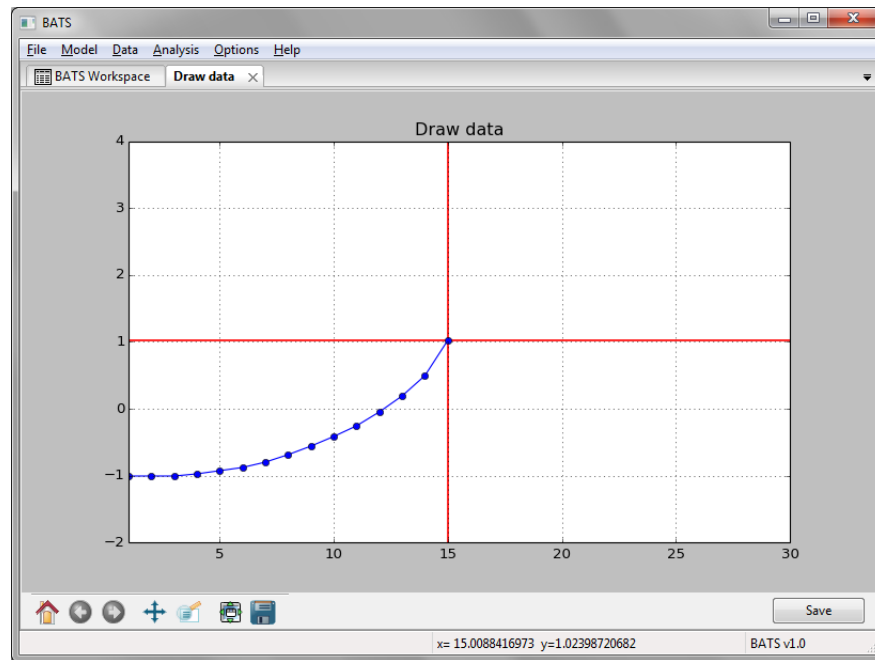
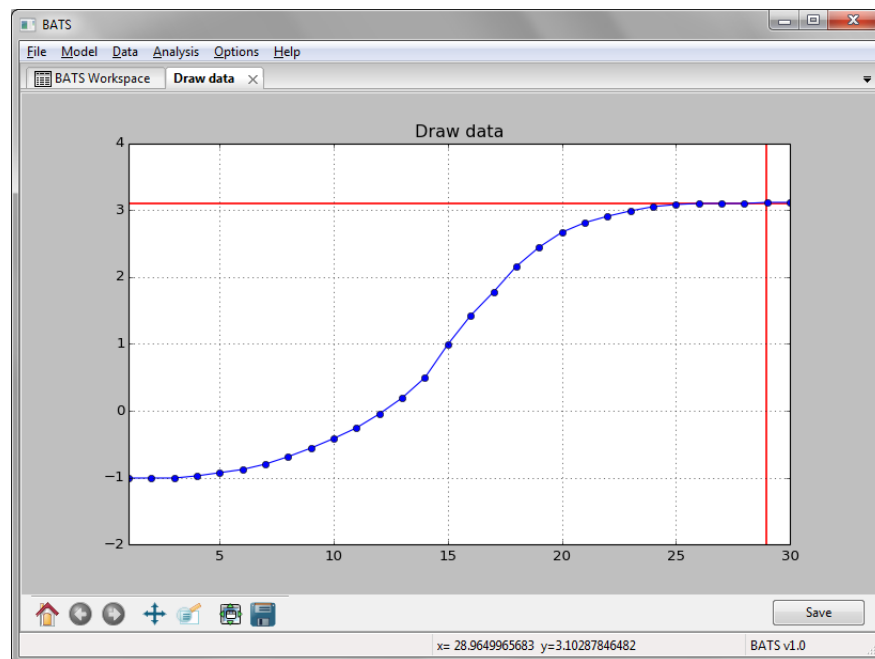


Figure 5.9. Input dialog for *Draw* tool.

In Figure 5.10, a usage example is illustrated in two phases. In this example an s-shaped growth pattern with specifications approximately initial level equals to -1 , inflection point is $(15, 1)$ and equilibrium level equals to 3 is drawn manually.



(a) Draw screen 1.



(b) Draw screen 2.

Figure 5.10. Illustration of *Draw* feature.

5.4.3. Split

Split function takes a data array as input and graphically displays the data to the user. User selects a value on the x-axis, and a red vertical line appears on the graph. When the user clicks *Save* button, left and right parts of the data are separated and stored as two new *DataItems* in the *Workspace*. While it can be used for other purposes this tool was specifically developed for splitting of transient and steady-state behaviors described in Section 4.1. As a result of that the default naming rule for the new parts are *InputData.L* for the left (early) portion and *InputData.R* for the right (later) portion. In the future work, automatically rather than manually specifying the splitting point, would increase the usefulness of this tool.

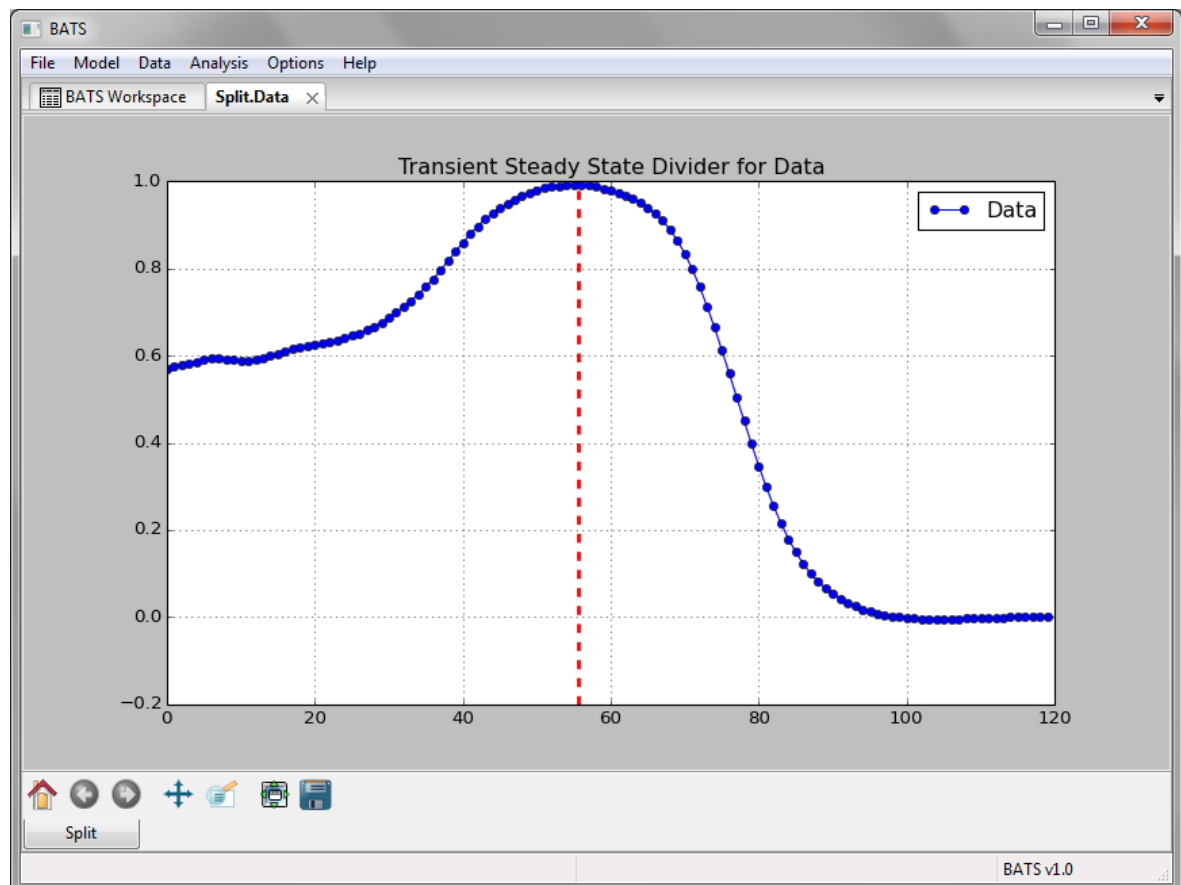


Figure 5.11. Screenshot for *Split* feature.

5.4.4. Crop

Crop feature is similar to *Split*, but it is a more generalized version of it. In this feature, the user selects two values on the x-axis, and clicking the Save button exports only the section of data that lies between those two limits. The new *DataItem* is named as *InputData.S*. This tool can be used for the general purposes of cropping and extracting data points from a bigger set.

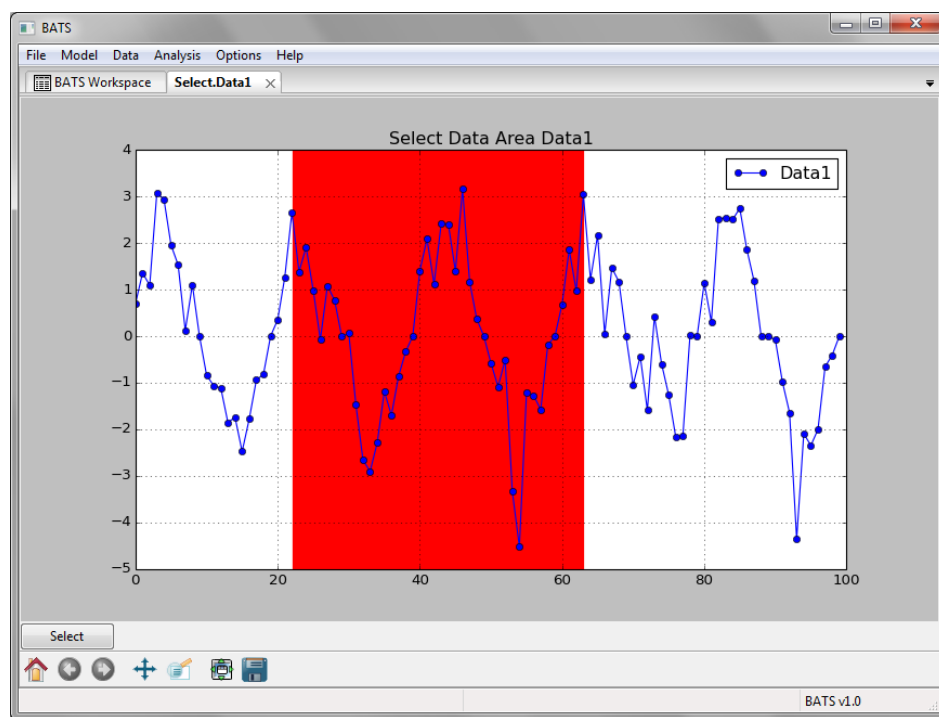


Figure 5.12. Screenshot for *Crop* feature.

5.4.5. Exponential Smoothing

Exponential Smoothing feature takes a data array as input and calculates the exponentially smoothed version of the input with a default “smoothing factor” (currently 0.5). The results are displayed on a graph consisting of two lines (original and smoothed). On the view there is a slider group. The user is allowed to change the smoothing factor either via slider or text input. As the smoothing factor changes the smoothed array is recalculated and the graph is updated simultaneously. After the user manually decides on the smoothing level and clicks Save button, the new smoothed data

array is exported to the Workspace as a new *DataItem* with name *InputData.ES*.

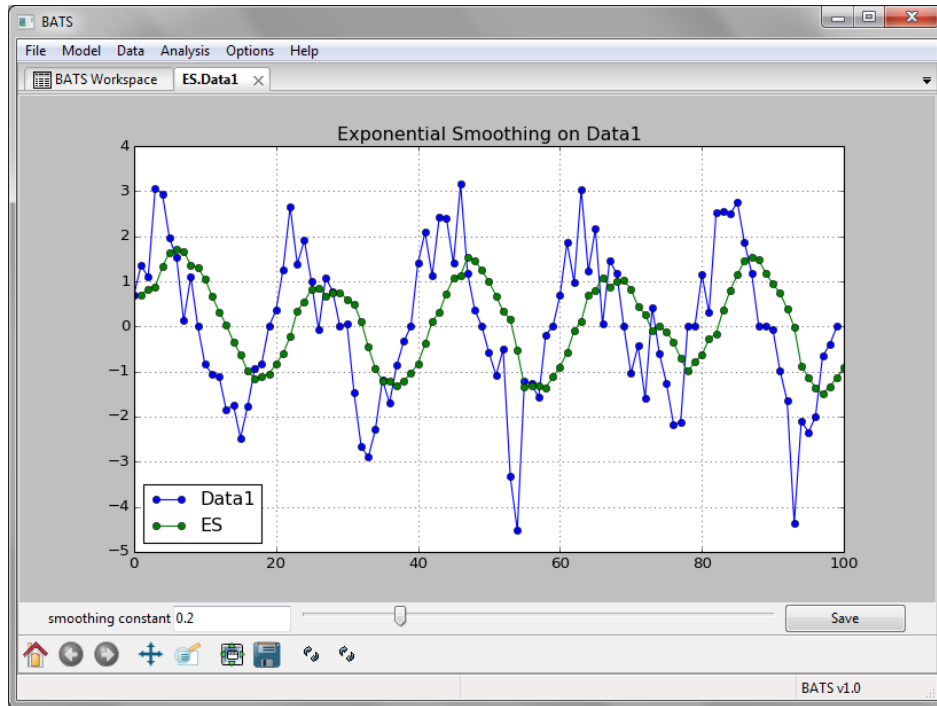


Figure 5.13. Screenshot for *Exponential Smoothing* feature.

5.4.6. Moving Average

The operation logic of *Moving Average* feature is similar to *Exponential Smoothing* feature except that the mathematical calculations are different. Consequently the parameter to be modified in the slider group is “terms”, which determines how many of the previous terms should be included in the filtered data. The name of the new data is set as *InputData.MA*.

5.5. Data Analysis Features

As a naming convention all user callable items that does a data related work is called a *Data Analysis Feature*. From a mathematical perspective, *autocorrelation* is a function and *pattern classification* is a method; however, in BATS the code that implements them are called *Autocorrelation Feature* and *Classification Feature* respectively.

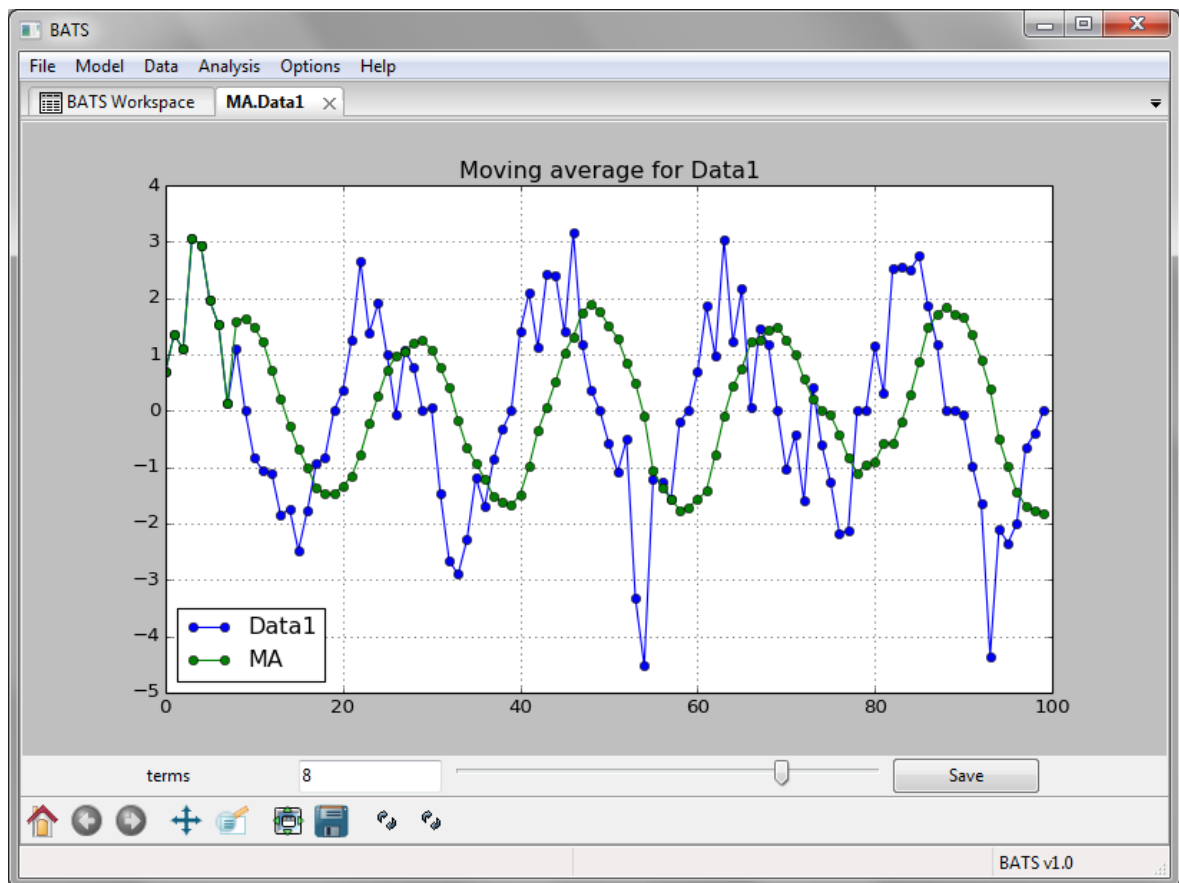


Figure 5.14. Screenshot for *Moving Average* feature.

Each feature is described in detail in the following pages. While each individual feature has special attributes, the following elements are common:

- (i) Start - Feature is called by the user
- (ii) Take inputs from the user, that can be fundamentally grouped into two:
 - one or more data arrays
 - additional function specific inputs like lags, smoothing coefficients, etc.
- (iii) Internally perform operations/calculations
- (iv) Update Logbook by writing the results as text
- (v) Create a new tab in the *Plotbook* and display results graphically
- (vi) If necessary include function specific information and user controls in the new tab on the display
- (vii) Wait for user commands. They can be one of the following command types:

- update graphical display
- export *DataItem* to the *Workspace*
- close the tab - Finish

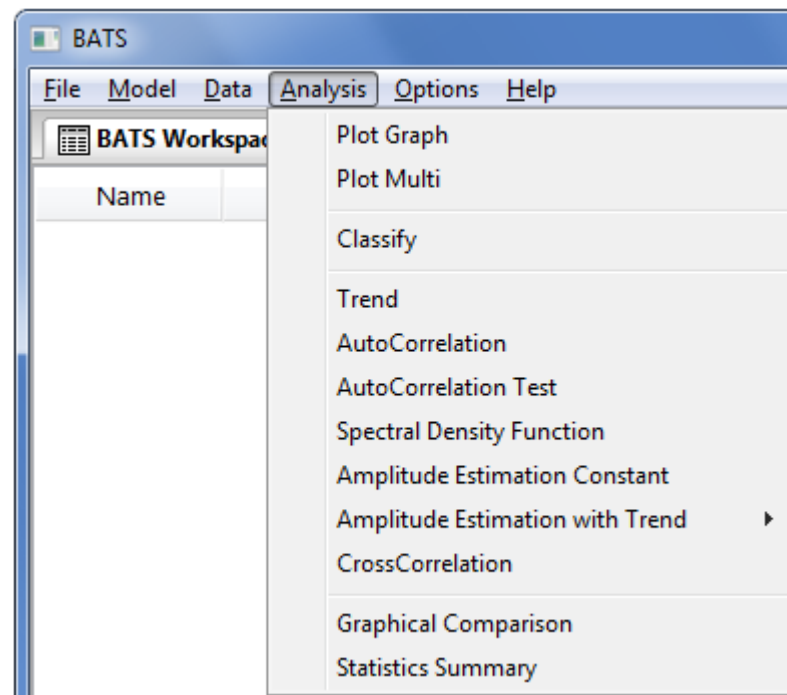


Figure 5.15. *Analysis* menu for data analysis features.

5.5.1. Plot

Utilization of user-friendly graphical inspection tools, plotting mechanisms, zooming capabilities are relevant to this study since they improve the graphical inspection and increase the user experience with the program.

Plot function takes a data array as input and plots it. Throughout this thesis the term “plot” refers to 2D graphics offered by Matplotlib, a library written in Python. Matplotlib is an excellent 2D graphics environment (Hunter, 2007). Not only it displays and exports publication-quality plots, it also has several very useful native plot tools; *Pan*, *Zoom*, and *Save*. In addition to these the plotting system allows the developers to include new tools.

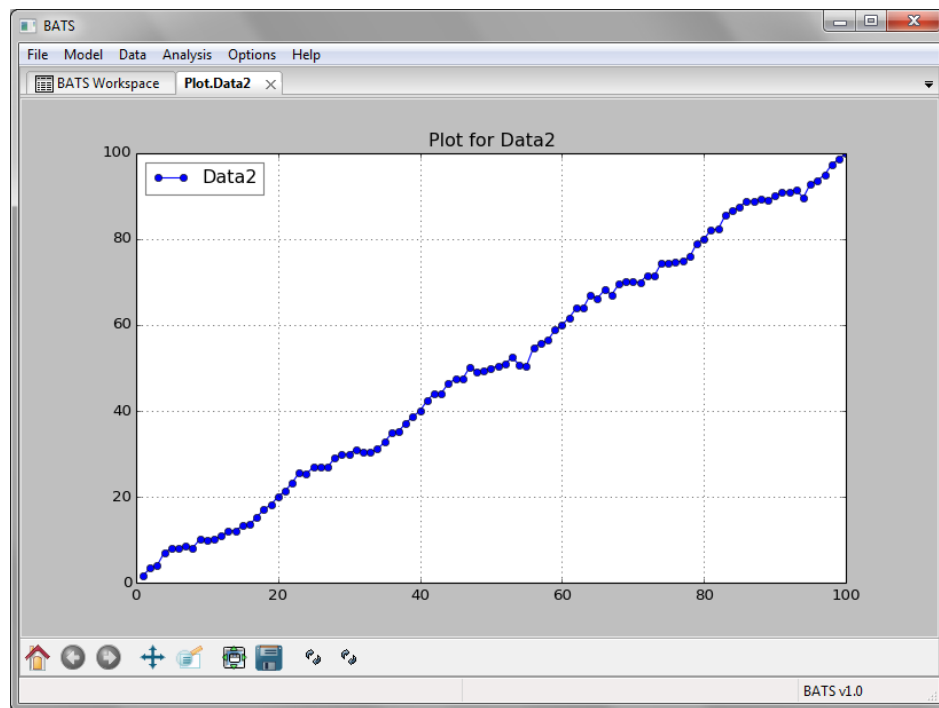


Figure 5.16. Screenshot for *Plot* feature.

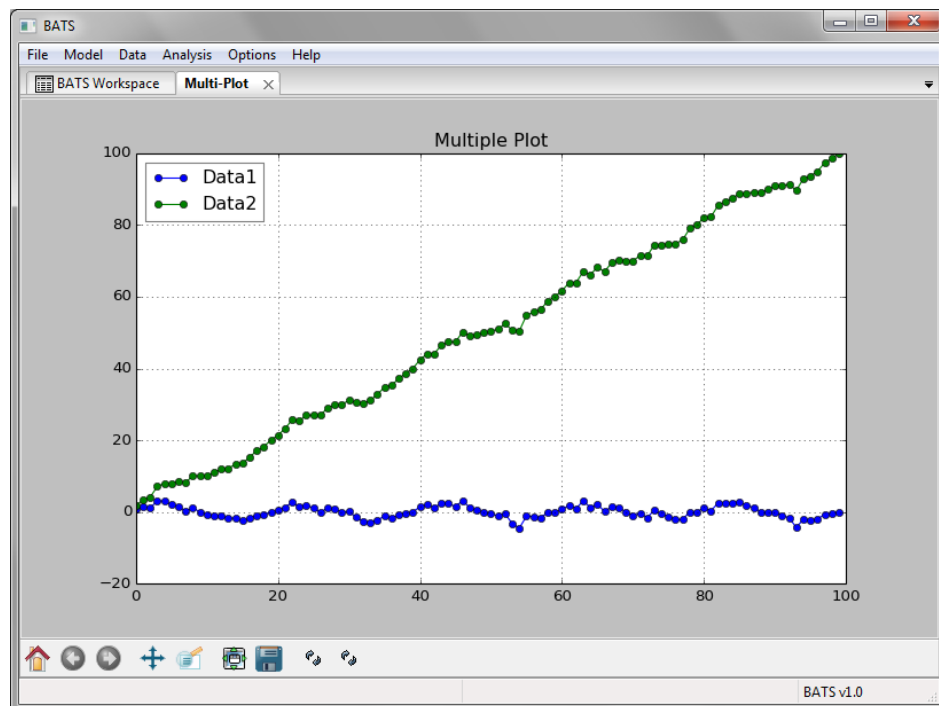


Figure 5.17. Screenshot for *Plot Multi* feature.

5.5.2. Trend

Trend feature takes a data array as input and calculates three types of trends of the input data. The trend types are linear, quadratic and exponential. For each trend operation the regression analysis is made, the coefficients are determined and a goodness of fit measure is provided. The results are plotted graphically along with the original input data to provide visual feedback to the user. User manually decides and selects which trend is appropriate for the input data. After the selection, the trend is removed from the input data and a new de-trended *DataItem* is added to the Workspace.

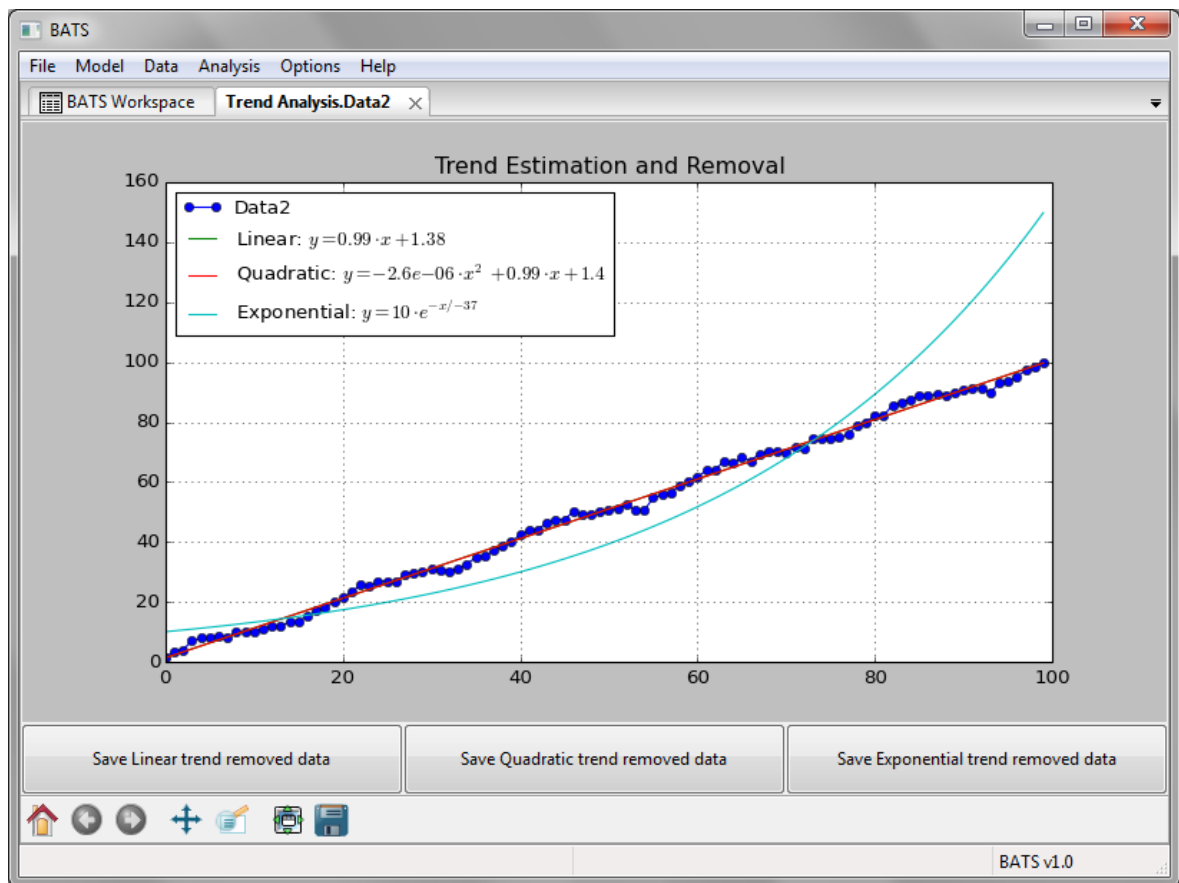


Figure 5.18. Screenshot for *Trend* feature.

5.5.3. Autocorrelation

Autocorrelation function takes a data array as input and returns the autocorrelation values at each lag. The lag are then displayed on a bar plot and written in the Logbook. This tool is useful for manually examining the data and determining fundamental period of the data if applicable.

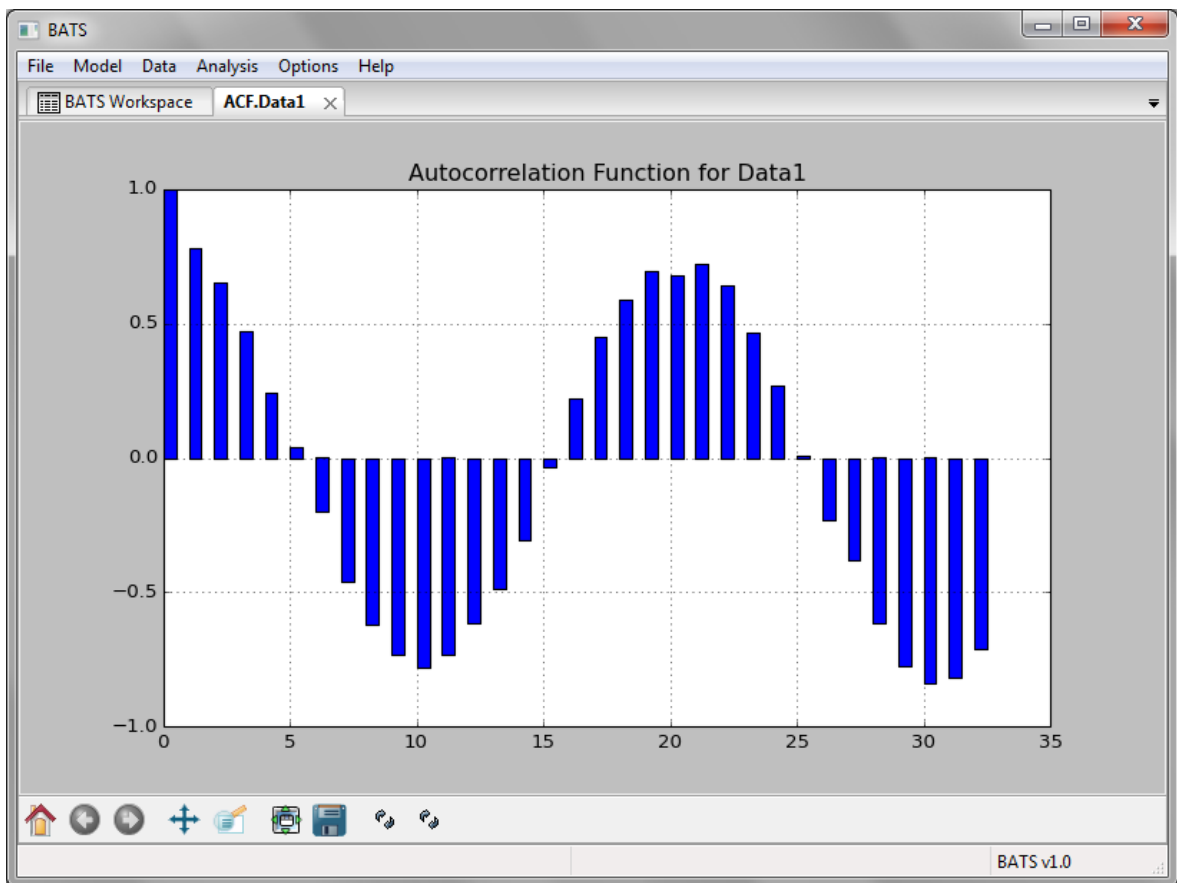


Figure 5.19. Screenshot for *Autocorrelation* feature.

5.5.4. Spectral Density

Spectral Density function takes a data array as input and returns the spectral densities at each lags. Similarly as in the Autocorrelation case the lags are displayed on a bar plot and written in the Logbook. This tool is useful for detecting significant periodicities in the data.

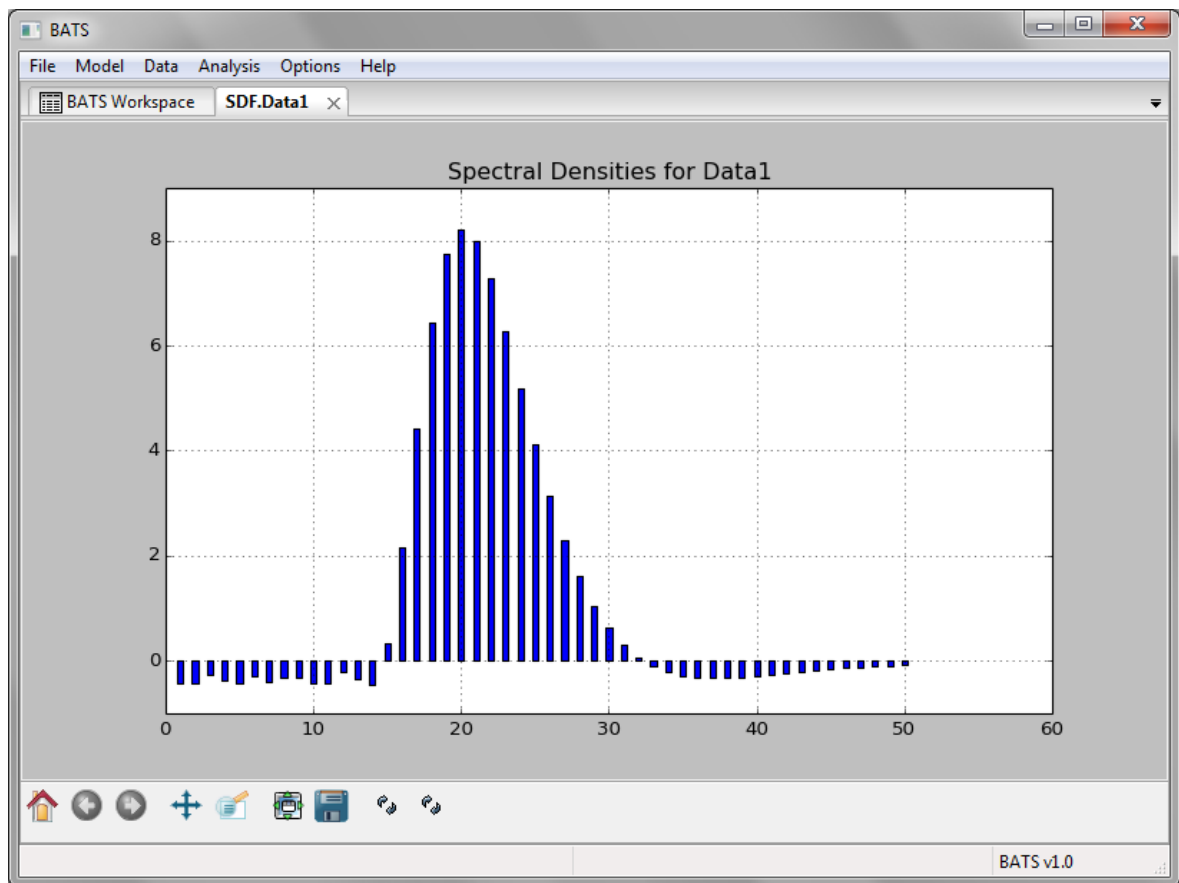


Figure 5.20. Screenshot for *Spectral Density* feature.

5.5.5. Amplitude Estimation - Constant

Amplitude Estimation - Constant feature takes a data array and an integer period as input, and perform trigonometric regression on the data using the given period as constant. The trigonometric function fitting method assumes that the sinusoidal oscillation is a good estimate for the input data. The sine wave is in the form of $\hat{y} = amplitude \cdot \sin(\frac{2\pi t}{period} + phase) + bias$. From this equation amplitude, phase and

bias are estimated and reported to the user in a slider group for each. If the values are changed by user, the graphical display gets updated. This tool allows the user to experiment with the fitted curve by manual modification and visual feedback. Also for numerical feedback, the goodness of fit is measured by using mean squared error formula and displayed on the legend.

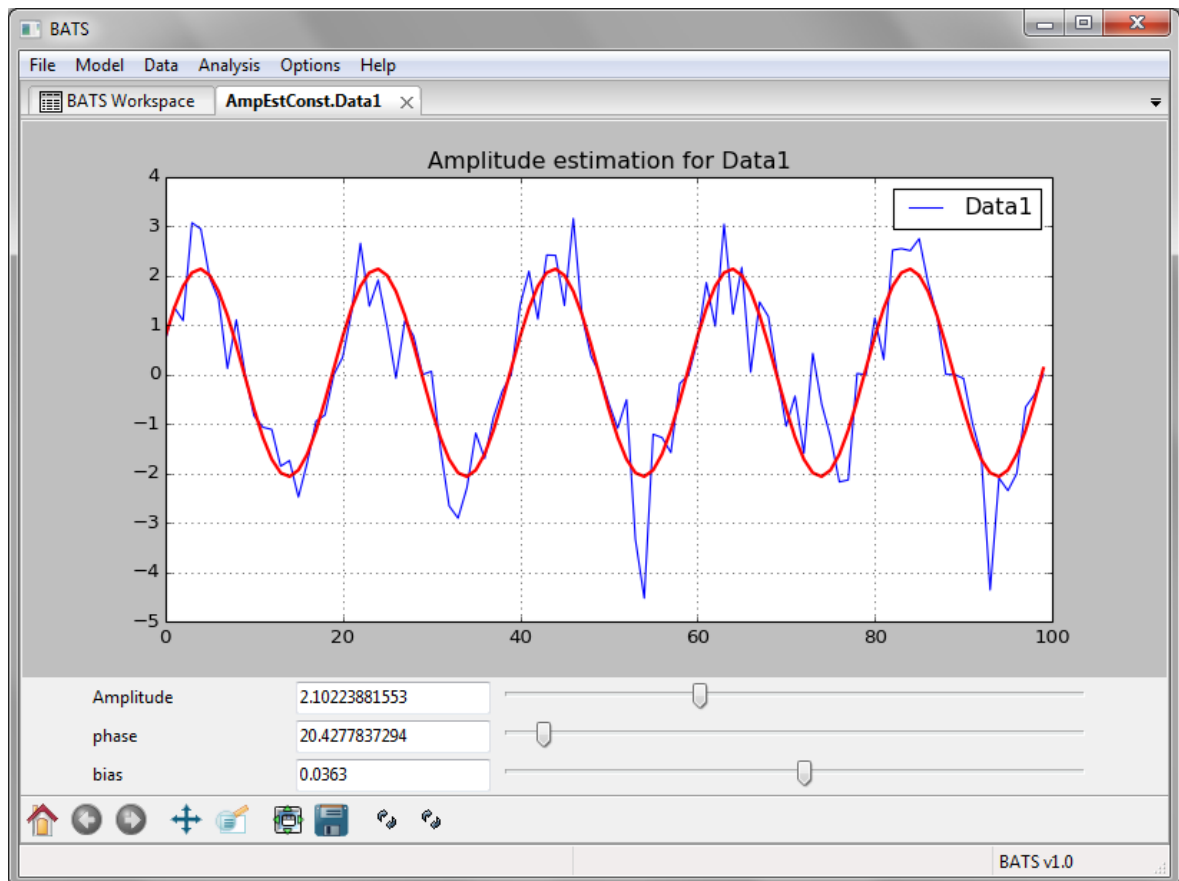


Figure 5.21. Screenshot for *Amplitude Estimation - Constant* feature.

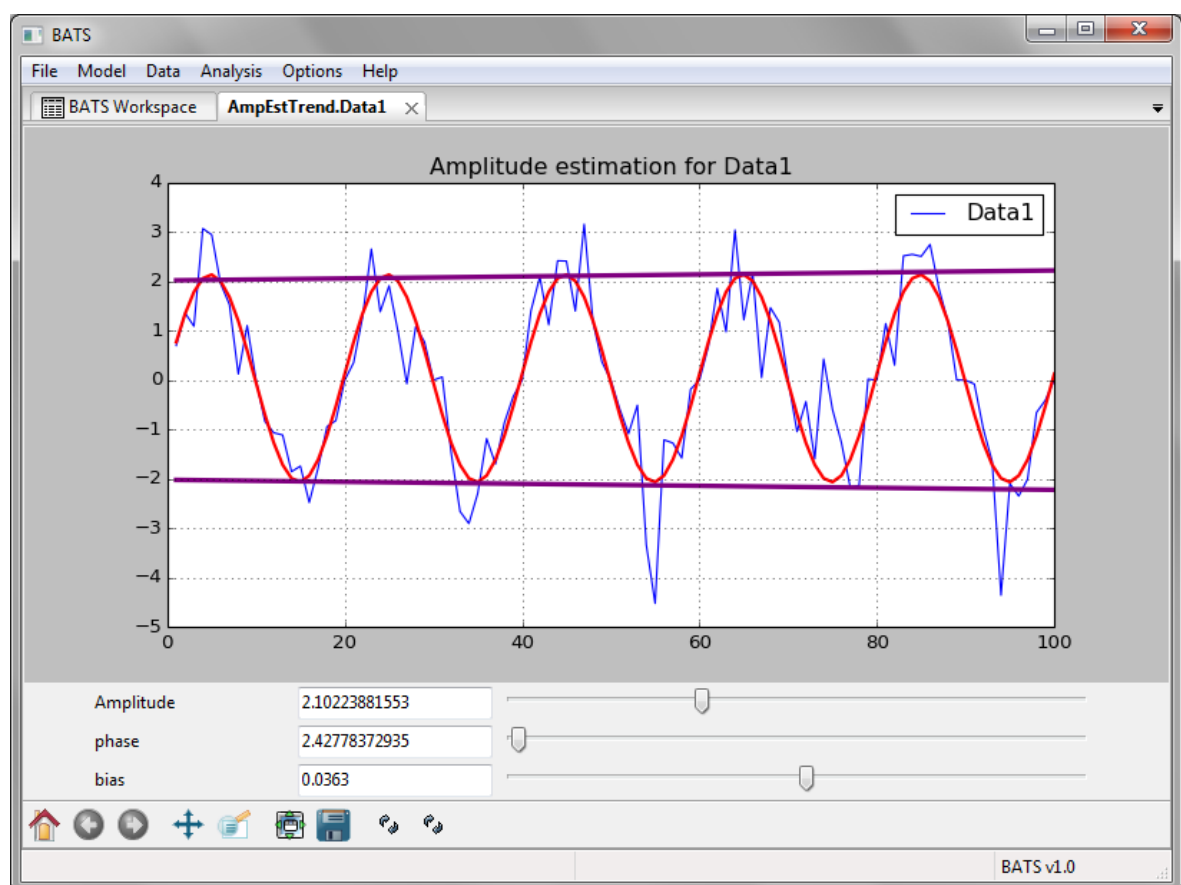


Figure 5.22. Screenshot for *Amplitude Estimation Trend* feature.

5.5.6. Cross Correlation

Cross Correlation feature takes two data arrays as input and returns their cross correlation values of each lag. The results are displayed on a bar plot and written in the *Logbook*.

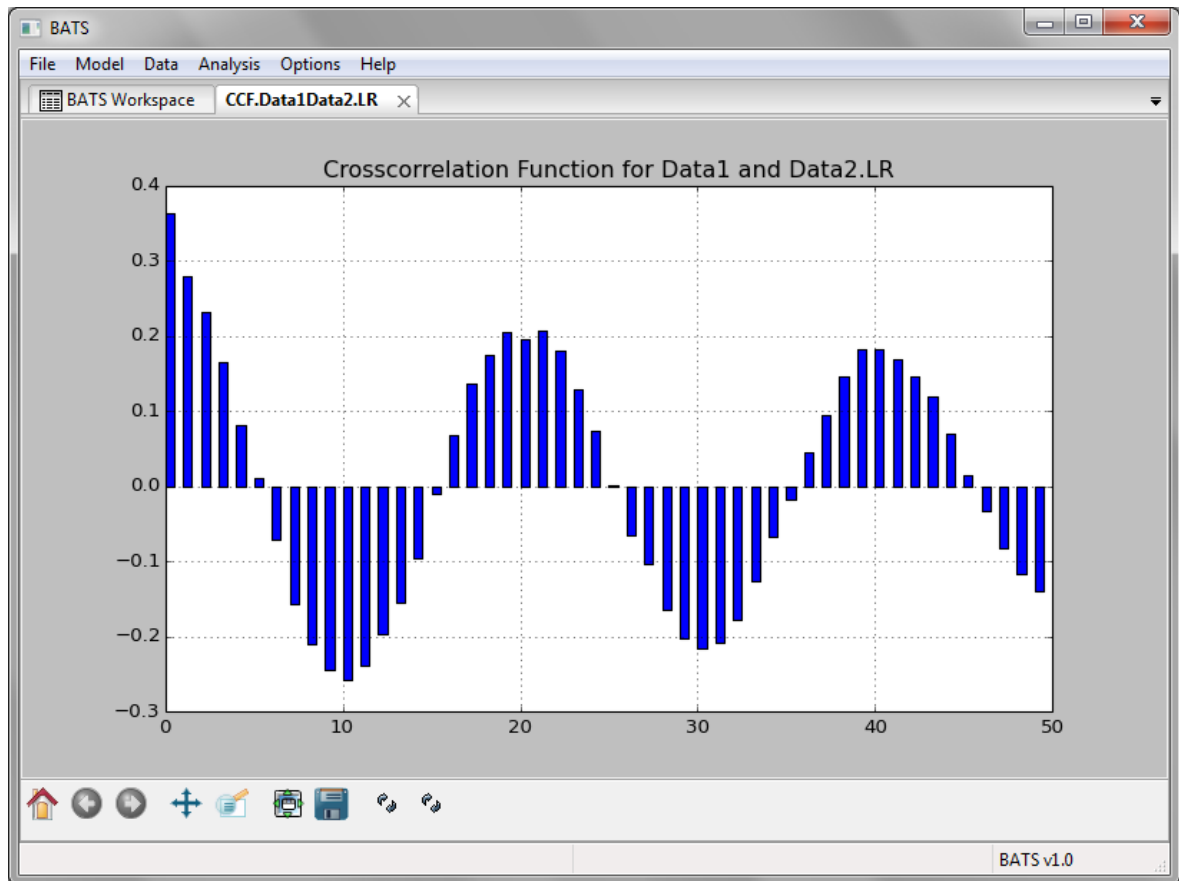


Figure 5.23. Screenshot for *Cross Correlation* feature.

5.5.7. Graphical Comparison

Graphical Comparison feature is useful for visually and manually comparing and analyzing two behavior patterns. It is specifically designed for transient behavior patterns, since most of the methods described in this section are not applicable to them. This feature implements the procedure described in Section 4.4. Comparing the key characteristics such as inflection points, equilibrium levels, maxima and minima is possible by using *Graphical Comparison*. There are four comparison alternatives:

- (i) X-point comparison (time point: t)
- (ii) Y-point comparison (level: y)
- (iii) X-interval comparison (duration: Δt)
- (iv) Y-interval distances comparison (distance or gap: Δy)

These four items are designed to span a wide range of custom key characteristics depending on the analysis purpose. For example, if the modeler is interested in inflection times, the *X-point comparison* option can be used; whereas if the modeler is interested in overall change in value of the state variable than *Y-interval comparison* option can be used.

Graphical Comparison feature takes two data arrays as inputs and creates a specialized *Tab* in the *Plotbook*. The tab consists of separate plots of the two input *DataItems*, a list, and a button labeled *Add Comparison*. When this button is clicked, a dialog box appears which lets the analyst select an option between the four comparison alternatives listed above. Then the user is guided for specifying the key points on the graphs in an order from left to right. After all the necessary points are determined the list is populated with a row consisting of the following information; editable comparison name field, value from the first input, value from the second input, difference between values and percent error. The analysis can be increased if necessary. Figure 5.24 illustrates a representative graphical comparison analysis done on two time-series.

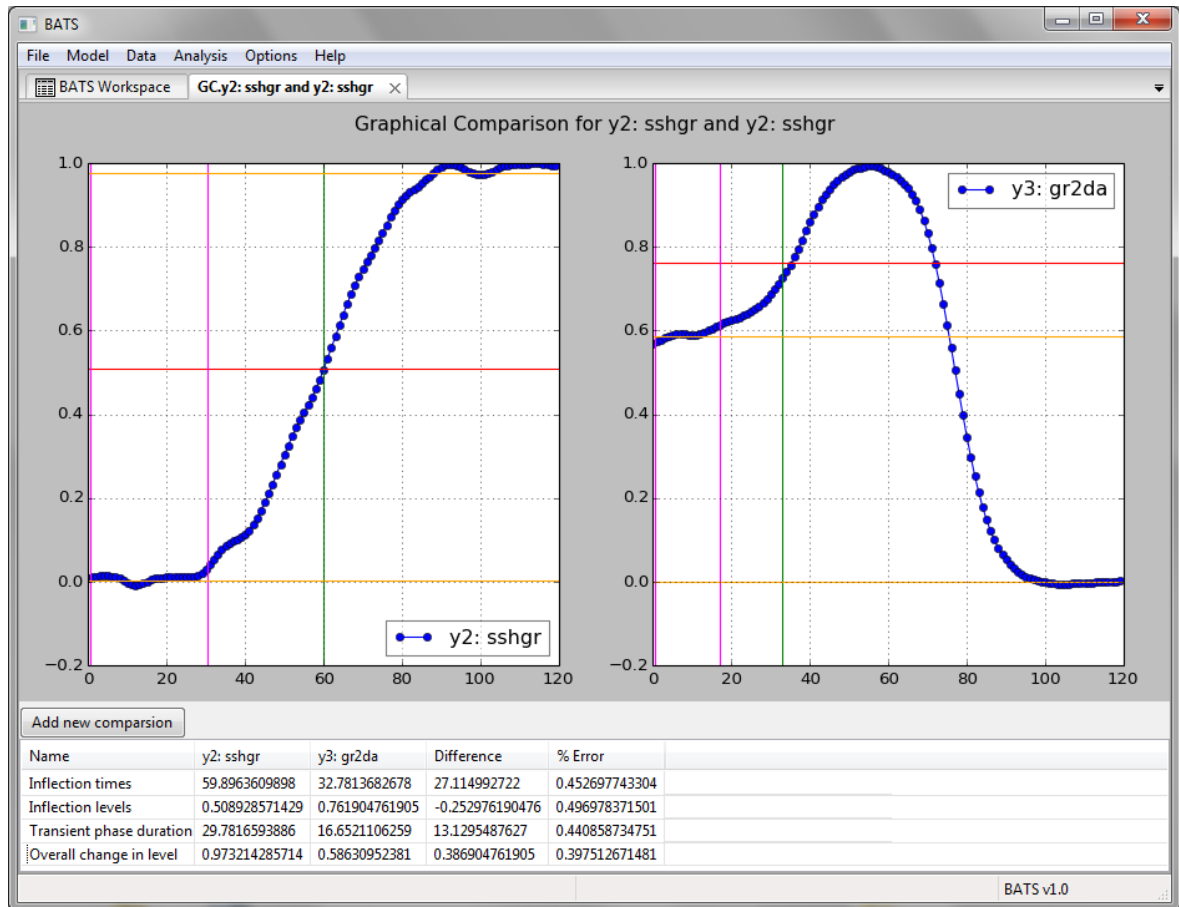


Figure 5.24. Screenshot for *Graphical Comparison* feature.

5.5.8. Classify

Classify feature takes a *DataItem* as input and returns the state-optimized likelihood values for each behavior pattern classes using ISTS Algorithm (See Section 4.2 for detailed information). The results are displayed to user in the *Logbook*. If none of the likelihood values are higher than the predefined acceptance level (currently set as -3.0), then an informative message is displayed on the *Logbook* as such: “*Behavior classification results are inconclusive. Try de-trending, smoothing or trimming the data.*”

5.6. Model Related Features

Model Related Features are functionalities that enable modeler/analyst to perform task-level automated analysis and communicate directly with simulation models developed on Vensim, one of the widely used model building software. In Figure 5.25, the screenshot from *Model* menu shows the available features implemented in BATS.

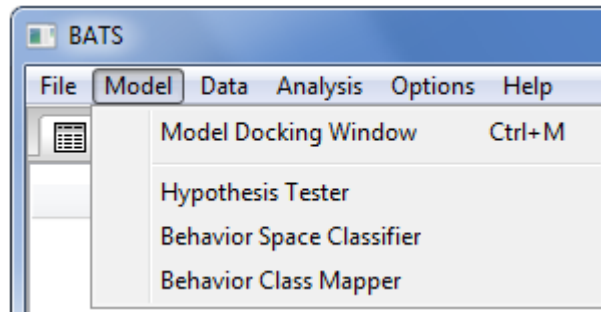


Figure 5.25. *Model* menu for model analysis features.

5.6.1. Model Docking Window

Model Docking Window is the most generic and basic functionality for communicating with the Vensim environment. It works with a Vensim model that is published as a packaged model (has *.vpm* extension). It is useful for all the usage purposes BATS aims at, namely indirect structure testing, sensitivity analysis, policy analyses, and especially behavior pattern tests. Essentially, this feature serves as a transferring mechanism of model outputs from models to *Workspace*, so that they can be stored as *DataItems* and be used in rich *Data Analysis Features*. To dock a model, refers to connecting BATS to Vensim via combination of *Vensim DLL* and Python's *ctypes* library.

The specialized *Tab* created by *Model Docking Window* can be seen in Figure 5.26. A general purpose work-flow for using this feature is given as follows:

- (i) The user clicks *Open Model Docking Window* under the *Model* menu
- (ii) A panel appears in the *Plotbook*. The panel includes a button for loading model file, a button for adding simulation runs, a list for storing and displaying the

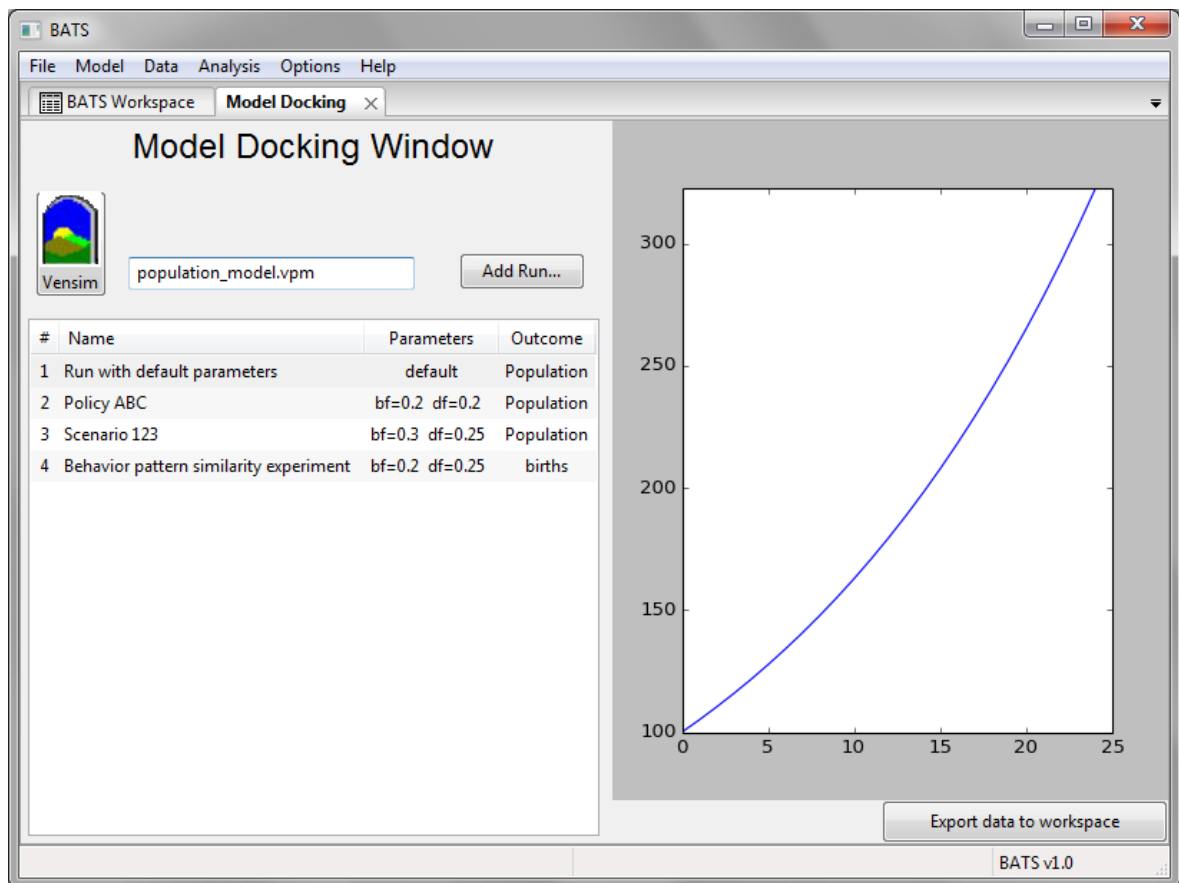


Figure 5.26. Screenshot for *Model Docking Window* feature.

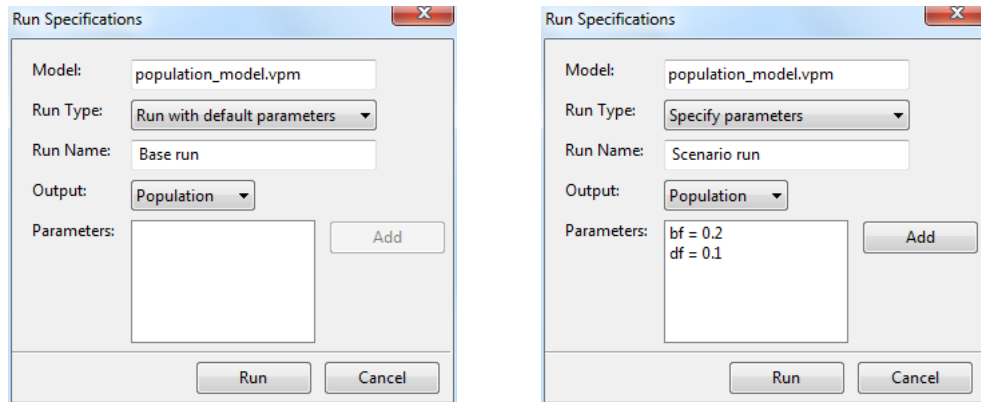
properties and results of simulation runs, a plot canvas for graphically displaying the results of the active simulation run, and a button for exporting the results of the selected experiment to the *Workspace* as a new *DataItem*.

- (iii) The user loads the Vensim model file (*MyFile.vpm*) using the file dialog box which appears after clicking *Vensim* button.
- (iv) The user adds a new experiment by clicking *Add Run...* button.
- (v) *Run Specifications*, a custom designed dialog box that is used for preparation of a new simulation run, appears on the screen, which can be seen in Figure 5.27.

This dialog has the following elements:

- *Model*: displays the name of the model for informative purposes.
- *Run Type*: gives the user a choice between two alternatives of run types: “*Run with default parameters*” or “*Specify parameters*”.
- *Run Name*: enables the user to set the name of the current run, some

descriptive examples can be given as: base run, reference behavior, case 1, optimistic/realistic/pessimistic scenario, extreme condition test - no deaths, extreme condition experiment - very high delays, behavior sensitivity run - productivity is doubled, no transfer policy, and so on.



(a) Run with default parameters

(b) Specify parameters

Figure 5.27. Two alternatives of *Run Specifications* dialog box.

- *Output*: lets the user select the outcome of interest. This is typically the “policy variable of importance” related to the purpose of the study.
 - *Parameters*: shows the parameters that are selected and modified by the user. The users sets a parameter using the *Add* button. The program shows the default value of the selected parameter for reference purposes and the user enters its new value for the new run. If the *Run Type* is set as “Run with default parameters” this section is disabled.
- (vi) The user ends the previous step by clicking *Run* and the program internally sets the specified parameters, runs the model, obtains the results of outcome variable, and stores it in a data array.
 - (vii) The list is populated with information related to this new run and the behavior of the outcome variable is plotted on the graph.
 - (viii) The user optionally exports the results of the outcome variable to the *Workspace* as a new *DataItem*, which can be later analyzed in depth using *Data Analysis Features*.
 - (ix) The user repeats steps 4-8 as many times as necessary.
 - (x) The user is allowed to return to the results of the previous experiments by clicking

on the corresponding row on the experiments list.

5.6.2. Hypothesis Tester

Hypothesis Tester feature in BATS is a specialized implementation of *Model Docking Window*. It is intended to be used for indirect structure tests that involve partial or full simulation of the model (Barlas, 1996). Typical usage mode can be for *extreme condition tests*. In this kind of tests the modeler/analyst checks whether the model is able to produce the hypothesized behavior pattern of the real system when these conditions are imposed on the model. By using *Hypothesis Tester* feature of BATS, these extreme conditions can be represented by setting extreme values to model parameters. In addition to that, this feature can be used for other in the validity tests such as behavior anomaly test and surprise behavior test.

The usage of the feature is intuitive. The user connects a model, sets extreme values to parameters, selects the outcome variable of interest and its hypothesized behavior pattern through the specialized dialog box illustrated in Figure 5.28. The behavior pattern classes are the ones that can be classified by ISTS Algorithm, and can be seen in Figure 4.2. The program gets the results of the simulation experiment and then calculates the behavior classification algorithm results using ISTS Algorithm. The class with the highest likelihood is found, and provided that it is above the pre-specified threshold value, it is determined as the behavior class of the simulation experiment. If the hypothesized behavior matches the simulated behavior then the program reports *PASSED*, else *FAILED*.

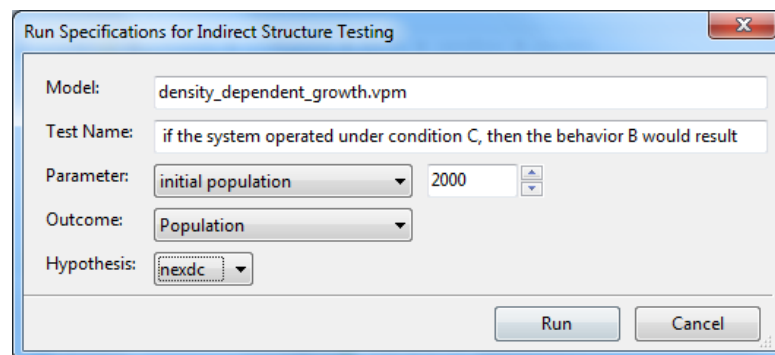


Figure 5.28. Hypothesis testing specifications dialog box.

The corresponding *Tab* of the *Hypothesis Tester* feature is shown in Figure 5.28. In this figure it could be seen that all the tests are stored throughout the session in order to provide quick summary to the modeler. The experiment results can optionally be exported to the *Workspace* as *DataItems* in order to further analyze the model output.

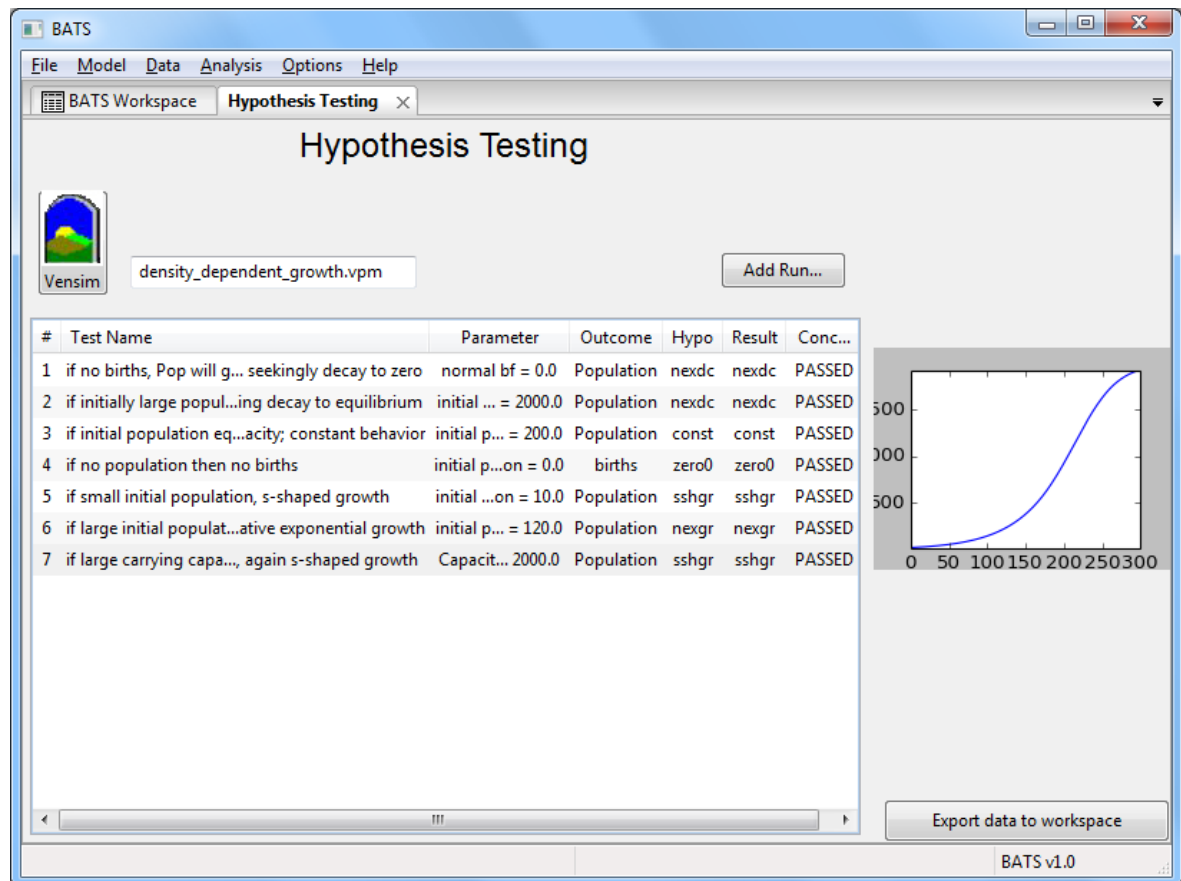


Figure 5.29. *Hypothesis Tester* results screen.

5.6.3. Behavior Space Classifier

Behavior Space Classifier feature is a highly automated analysis tool in BATS. This tool provides automated behavior pattern recognition on model outputs directly from Vensim model files (.vpm) and useful for purposes such as sensitivity analysis, model calibration and policy analysis. Behavior space exploration is a rare application in output analysis. Currently, by using this feature, it is possible to search the behavior sensitivity of a model by complete enumeration for 2 parameters of uncertainty, that is performing behavior pattern sensitivity with respect to changes in parameters.

Model file, output variable, output measure, names and ranges for uncertainty parameters are collected as inputs from the user. Output measure is the behavior class pattern with the maximum likelihood (see Figure 5.31). The likelihood values are calculated using the ISTS algorithm.

Behavior Space Classifier produces the two figures as illustrated in Figure 5.31. In this *Tab* there are two plot areas. The top area is a classifier grid populated with respect to sensitivity values of parameter 1 and parameter 2. The cells on the grid are filled with labels of the behavior pattern classes with the highest likelihood. In this final plot, a coloring system is embedded in order to improve visual classification. As the user clicks on the classifier grid, the lower plot changes to show the output of the corresponding run. Furthermore, detailed classification results for all twenty-five behavior patterns are displayed in the *Logbook*.

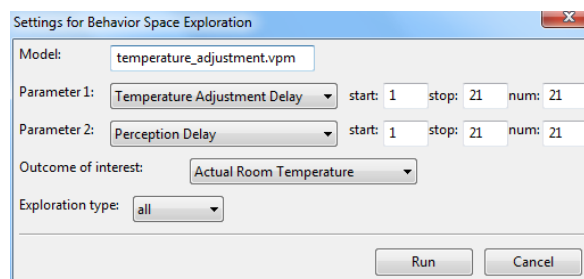


Figure 5.30. Settings dialog box for *Behavior Space Classifier*.

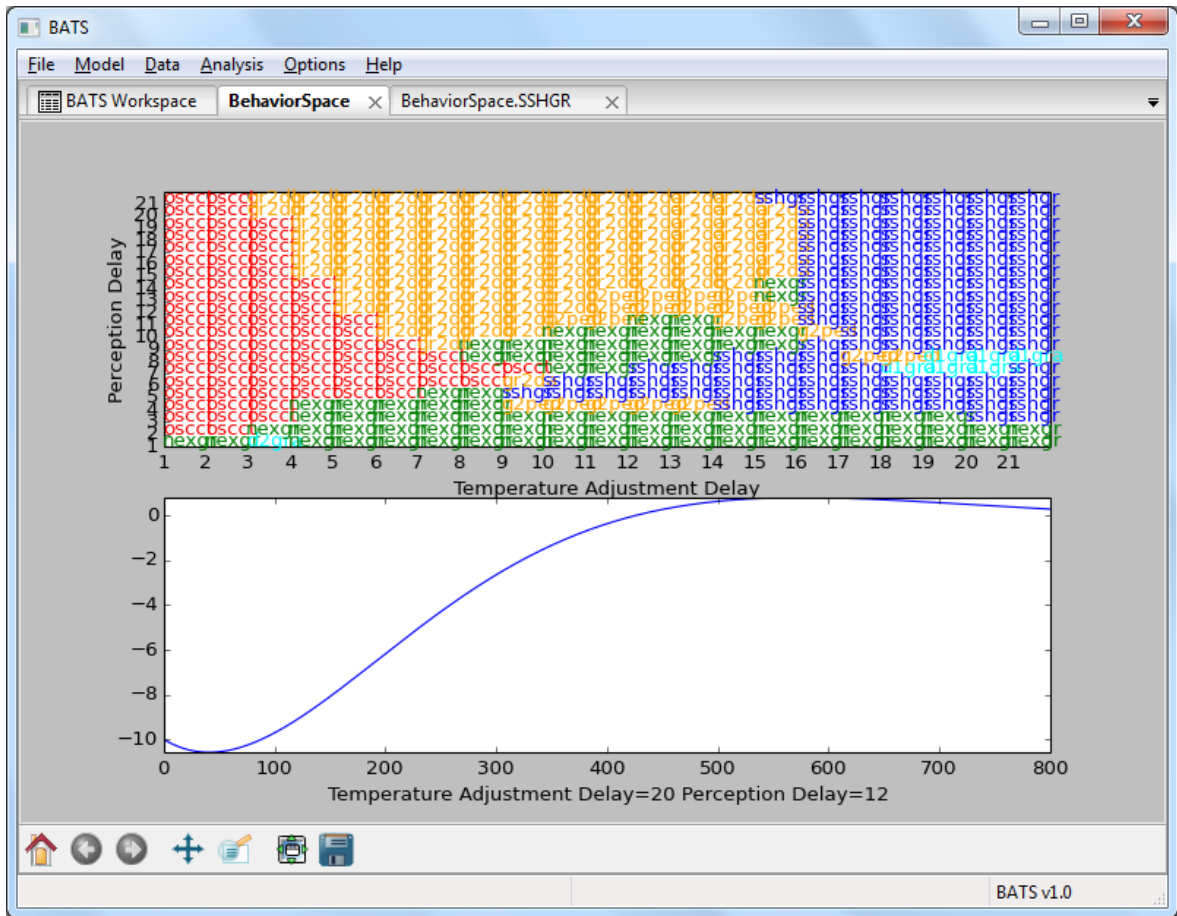


Figure 5.31. Screenshot for *Behavior Space Classifier* results.

5.6.4. Behavior Class Mapper

Behavior Class Mapper feature of BATS is a method for assessing changes in the behavior mode of the model as a result of changes in its parameters. The preparation step and the results *Tab* is similar to those of *Behavior Space Classifier*, but the output metrics and information is different. In the preparation step the user chooses a specific behavior pattern such as *sshgr* (s-shaped growth), as well as parameters of uncertainty and their ranges and output variable. The resulting *Tab* can be seen in Figure 5.33.

In Figure 5.33 each point on the plot corresponds to one simulation run and for each run model output is obtained from Vensim. For each model output ISTS Algorithm is called and likelihood value for the pre-specified behavior class is plotted on the graph. This process is repeated for all parameter combinations. The intermediary

parameter values are interpolated in order to obtain a contour plot with respect to parameter 1 and parameter 2. In this contour plot the warmer colors indicate higher likelihoods and vice versa. The detailed likelihood results for all 25 behavior patterns are displayed in the *Logbook*.

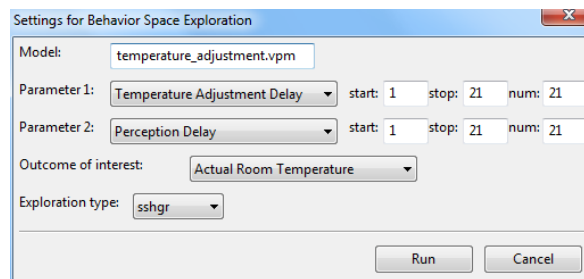


Figure 5.32. Settings dialog box for *Behavior Class Mapper*.

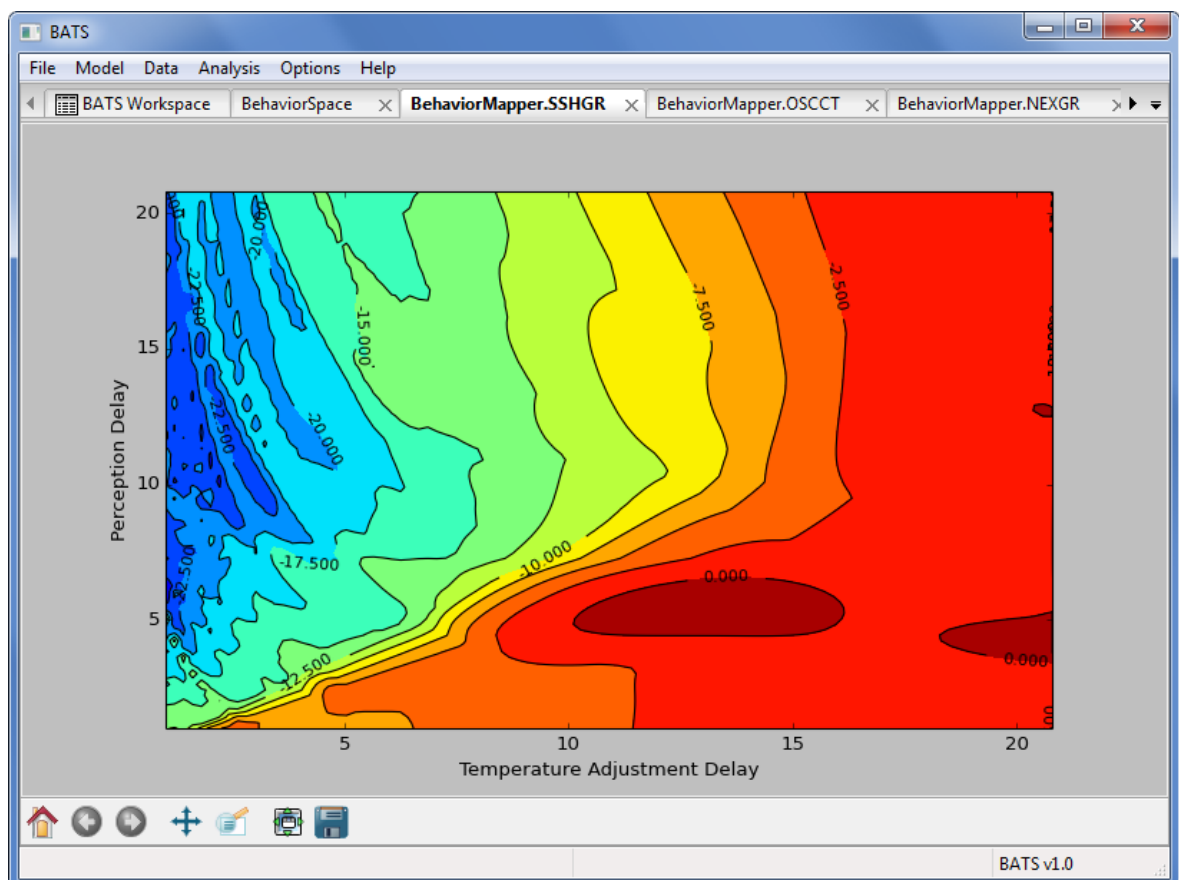


Figure 5.33. Screenshot for *Behavior Class Mapper* (results for s-shaped growth pattern).

6. DEMONSTRATION AND USAGE MODES OF BATS

In this chapter possible usage modes of BATS will be demonstrated. In order to span as many functionalities as possible, alternative file loading and model connection techniques are demonstrated.

6.1. Extreme Condition Testing

6.1.1. Test Model: Density-Dependent Growth Model

Extreme condition tests are behavior oriented structure tests and can help the modeler discover potential structure flaws. Not necessarily all of the extreme conditions have to be empirically observed in real life, they also apply to theoretical evidence and knowledge. In order to demonstrate this usage of BATS, we are going to use the *density-dependent growth* model. Also see (Barlas, 2002, p. 22) and (Sterman, 2000, p. 118). In this model birth rate is affected by *crowding* (or *carrying capacity*). As the crowding increases the birth rate decreases. This effect is represented by an *auxiliary with lookup* variable. The constant death rate decreases the population proportionally. Since the model is a first order system, it does not oscillate. If population becomes equal to capacity, the birth fraction becomes equal to death fraction, so that the population becomes stable. The stock-flow diagram can be seen in Figure 6.1. The system can be mathematically represented as:

$$\frac{dPopulation(t)}{dt} = births(t) - deaths(t)$$

$$births(t) = Population(t) \times bf(t), \text{ where } bf(t) = bf_{norm} \times f(Crowding)$$

$$Crowding = Population/Capacity \text{ and } bf_{norm} \text{ is constant}$$

$$deaths(t) = Population(t) \times df, \text{ where } df \text{ is constant}$$

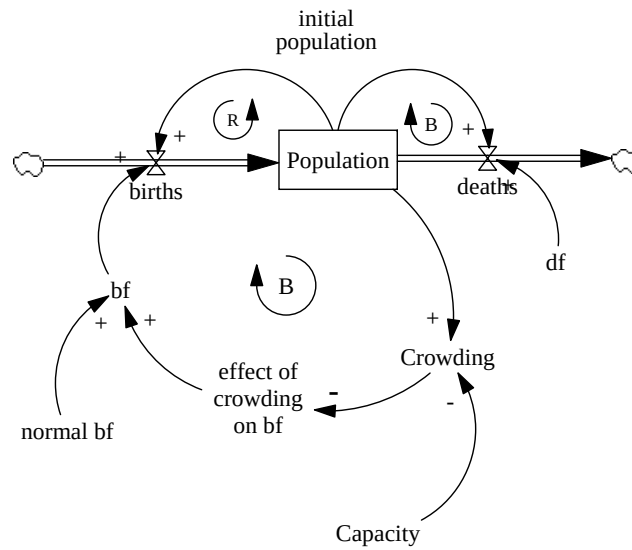


Figure 6.1. Stock-flow diagram of the density-dependent growth model.

6.1.2. General Usage Procedure

Step 1: External preparation of the Vensim model. In this step the model *density_dependent_growth.mdl* must be saved as “.vpm” file. In order to do this, the user chooses from File-Publish menu in Vensim and then saves the model as *density_dependent_growth.vpm*.

Step 2: The user chooses from *Hypothesis Tester* item under Model menu. The initial screen contains a blank list, a blank plot and buttons. The user connects a model to BATS by clicking *Connect Model*. After a successful connection to the model the text field is updated and shows the name of the model file. At this step BATS internally gathers the variable names of the model in order to show later to the user.

Step 3: In order to perform a new test, the user clicks *Add Run...* button. After that, a dialog box opens. In the dialog box, one optional information the test name, and four required information; parameter name, parameter value, outcome variable, hypothesized behavior are filled by the user. The default value for the test name is “*if the system operated under condition C, then the behavior B would result*”. The user clicks *Run* button and the model is simulated with the given settings.

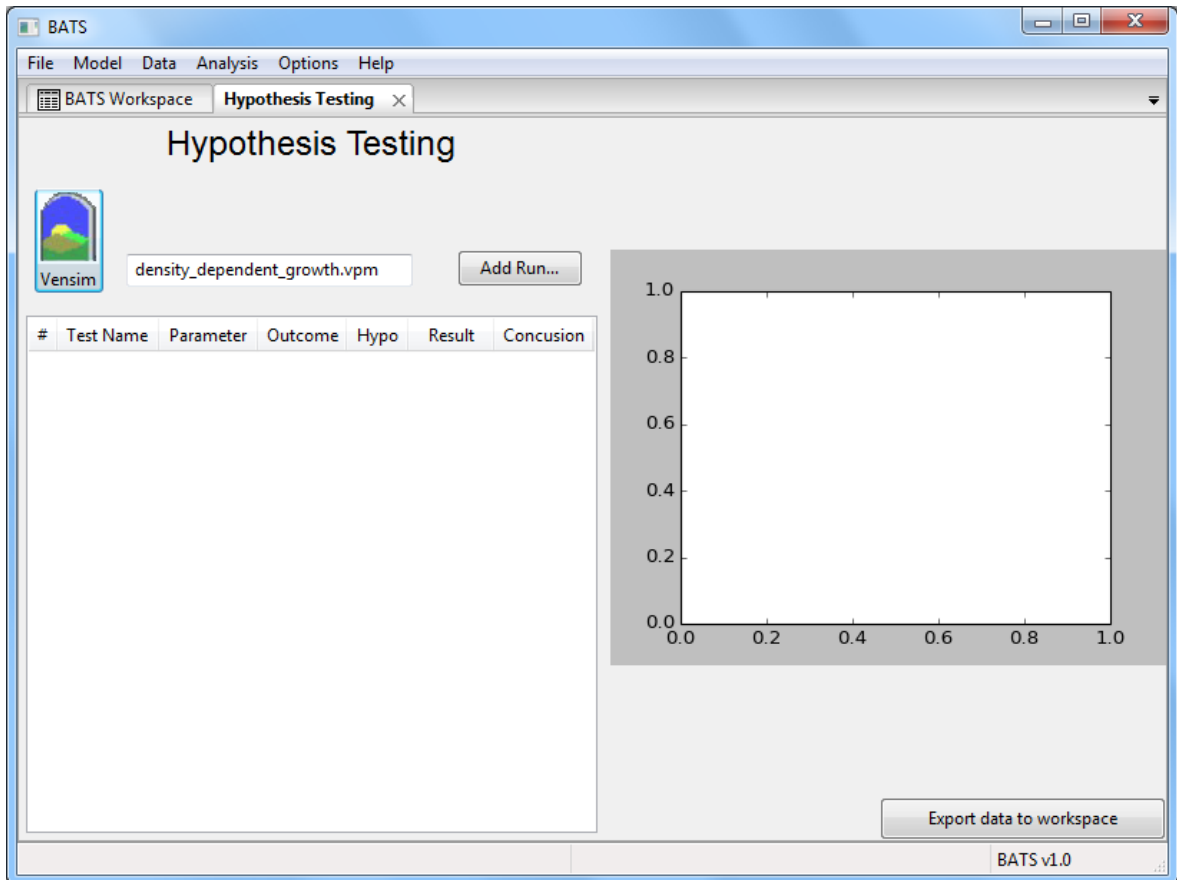


Figure 6.2. Preparation of the model for *Hypothesis Tester* feature.

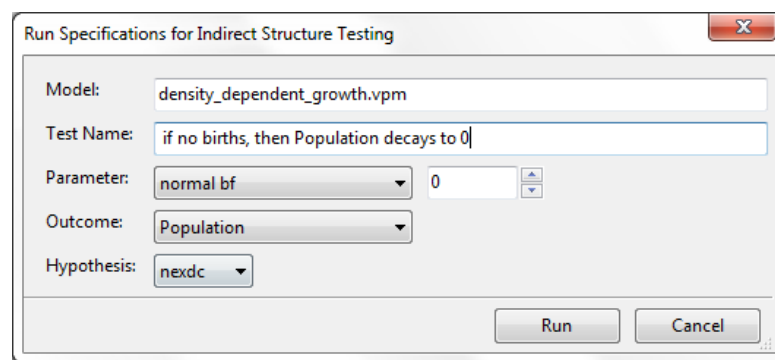
Step 4: The likelihood outcome variable to belong to each generic dynamic patterns are calculated by ISTS Algorithm (see Figure 4.2 and Section 4.2). The class with the highest likelihood is identified as the behavior class of the model output. The program makes a True/False comparison and concludes as either *PASSED* or *FAILED*.

Step 5: The list and the plot are updated. The user can export the data of the outcome variable into the Workspace in order to conduct further analysis on the obtained output, by clicking *Export Data to Workspace* button.

These are the general steps of the *Hypothesis Tester* feature of BATS. Step 3 to 5 can be repeated as many times as desired each time a new row gets added to the results list. The previous results can be revisited by double-clicking on the corresponding row. Final list presents an overview to the user at the end of the analysis session.

6.1.3. Tests and Results

Hypothesis 1: If for some reason there happens to be no births in the system then *Population* would gradually erode to extinction. Representation in the model: Set $normal_bf = 0$ and hypothesize the behavior of *Population* as negative exponential decline (nexdc) as in Figure 6.3. The test is *PASSED* with likelihood value of 0.3997. The results can be observed in Figure 6.4.



Run Specifications for Indirect Structure Testing

Model: density_dependent_growth.vpm

Test Name: if no births, then Population decays to 0

Parameter: normal bf 0

Outcome: Population

Hypothesis: nexdc

Run Cancel

Figure 6.3. Test specifications for the first hypothesis.

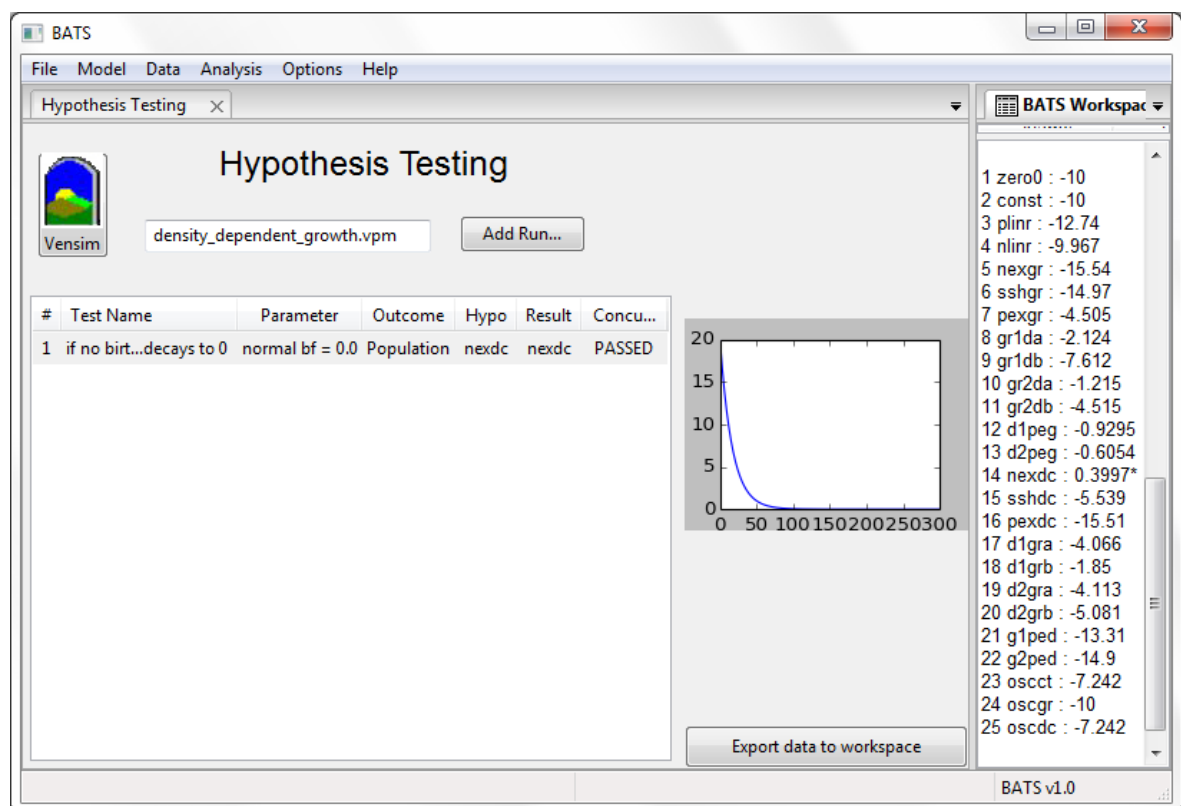
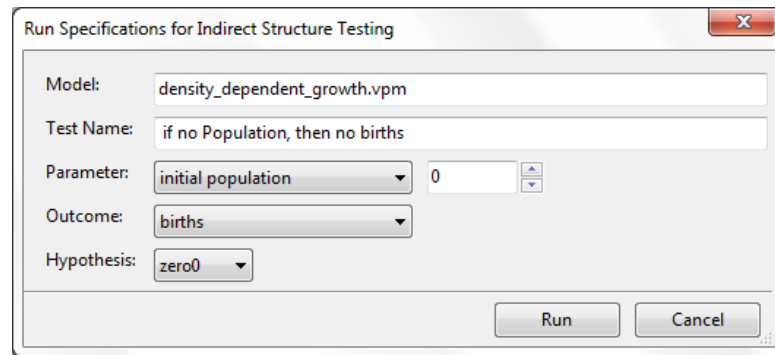


Figure 6.4. Test results for the first hypothesis.

Hypothesis 2: If there are no *Population* in the system, then there would be no births. Representation in the model: Set *initial_population* = 0 and hypothesize the behavior of flow variable *births* as constant at zero (zero0) as in Figure 6.5.



Run Specifications for Indirect Structure Testing

Model: density_dependent_growth.vpm

Test Name: if no Population, then no births

Parameter: initial population 0

Outcome: births

Hypothesis: zero0

Run Cancel

Figure 6.5. Test specifications for the second hypothesis.

The test is *PASSED* with likelihood value of 0. The results can be observed in Figure 6.6.

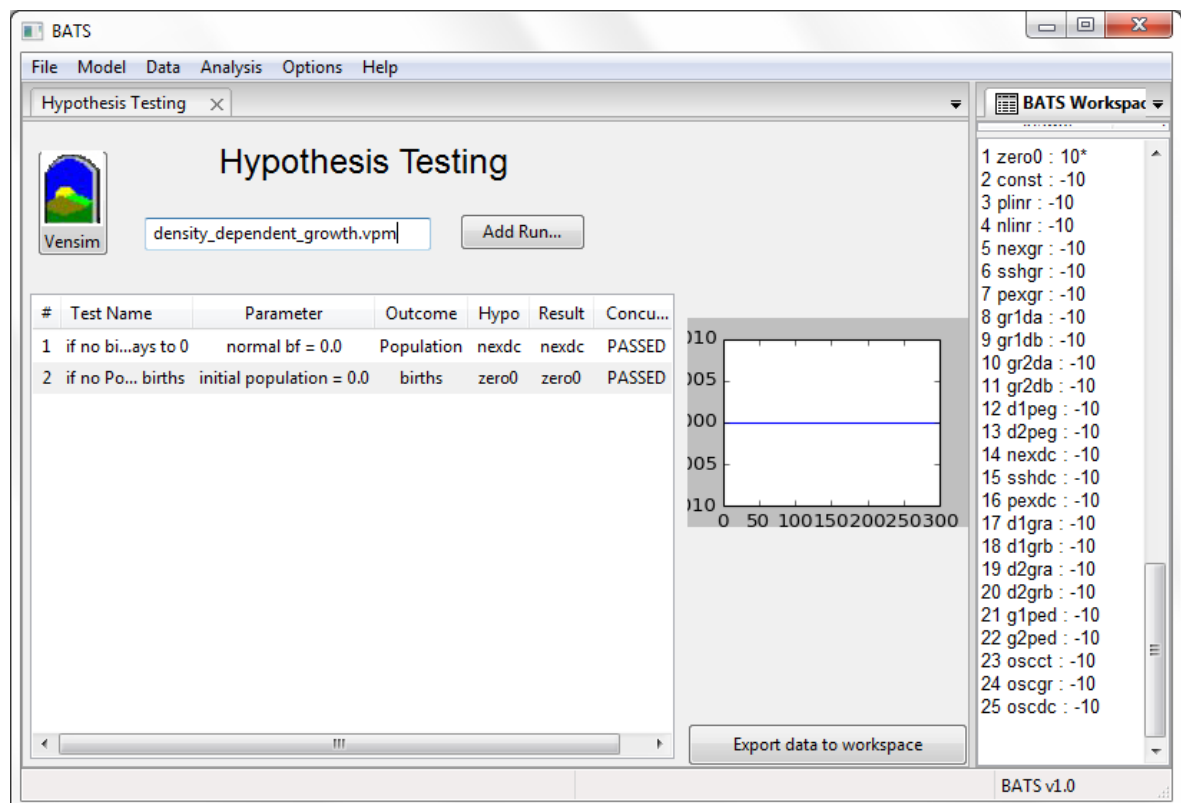
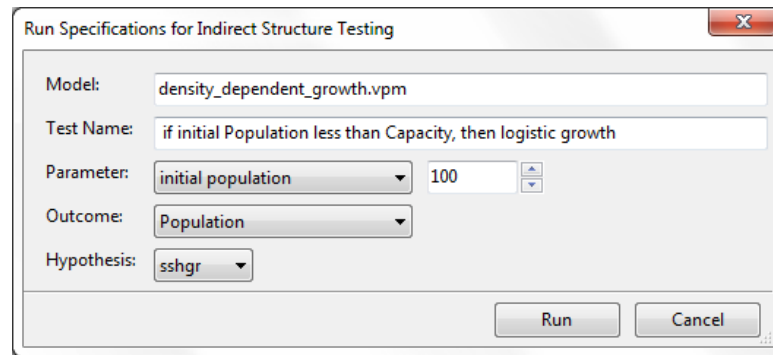


Figure 6.6. Test results for the second hypothesis.

Hypothesis 3: If the initial population is less than the carrying capacity then it will stay at the equilibrium. Representation in the model: Set *initial_population* = 100 and hypothesize the behavior of *Population* as s-shaped growth (sshgr) as in Figure 6.7.



Run Specifications for Indirect Structure Testing

Model: density_dependent_growth.vpm

Test Name: if initial Population less than Capacity, then logistic growth

Parameter: initial population 100

Outcome: Population

Hypothesis: sshgr

Run Cancel

Figure 6.7. Test specifications for the third hypothesis.

The test is *FAILED* with likelihood value of 1. This is due to the fact that input value for *initial_population* = 100 is above the inflection level for s-shaped growth. The results can be observed in Figure 6.8.

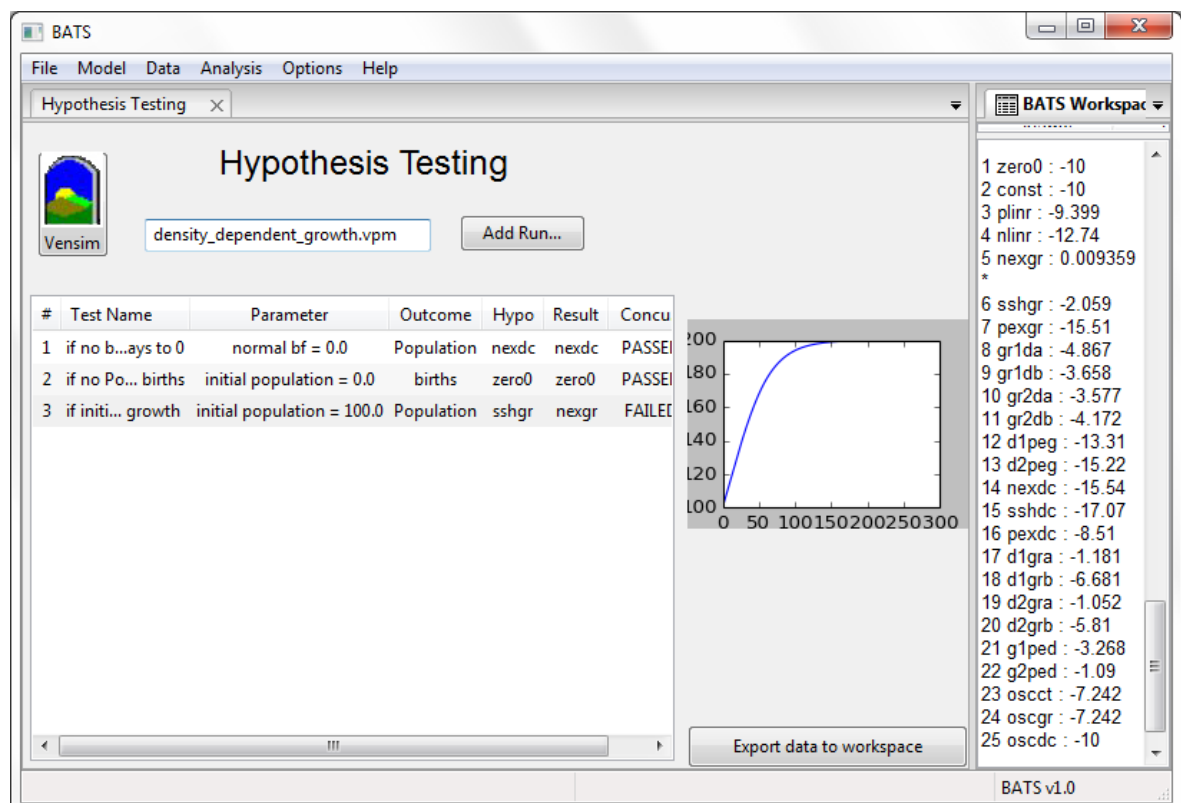


Figure 6.8. Test results for the third hypothesis.

This model analysis feature of BATS not only enables automated behavior analysis, but at the same time encourages more experiments being held and reported in an organized manner. The view can be customized as can be seen in Figure 6.9, and the individual results of the experiments can be exported to the *Workspace* in order to be used in data analysis features. In this figure three more extreme conditions tests are conducted in addition to previous three examples.

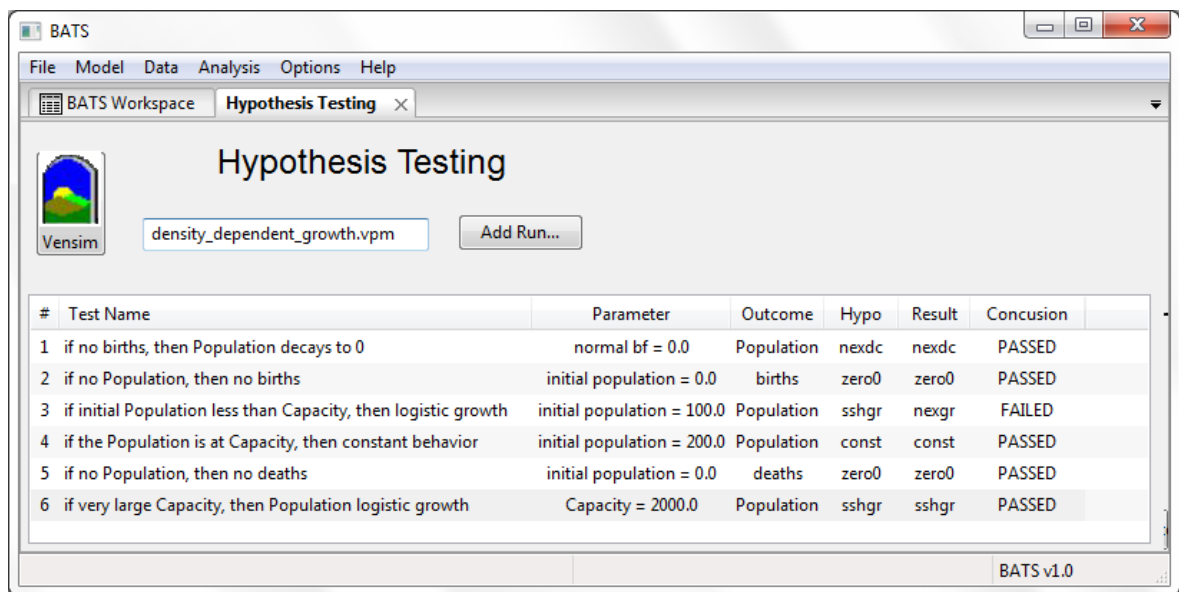


Figure 6.9. Screens from the end of the analysis session.

6.2. Behavior Pattern Tests

In principle, behavior pattern tests are done after confidence on model structure has been established. If these test fail, the model revision is generally done on a parameter/input re-calibration and further parameter estimation as opposed to structural changes (Barlas, 1996).

In this demonstration two data series are compared. Firstly Barlas (1996)'s *Multi-Step Procedure* for steady-state behavior pattern validity is fully implemented, and secondly, *Graphical Inspection* on transient behavior pattern validity is demonstrated. The *Multi-Step Procedure* involves statistical measures applied in a logical order. For this demonstration one data series is loaded from an Excel file, and one from a text file. Also in order to show print-quality image exporting capability of BATS, all of the figures in this section are taken and used directly from the software.

Step 0: Load data from files and prepare for Multi-Step Procedure. Real data is loaded from *MyRealData.txt* an Excel spreadsheet and model output is loaded from *MyModel.xlsx* a txt file. *Workspace* is updated with these new *DataItems*. Then the user modifies the original data series' names as *Actual* and *Model* accordingly. From Figure it can be observed that the two time-series involve a transient phase that have durations of approximately one cycle. In order to continue the procedure, the transient and steady-state parts should to be separated. This is achieved using *Split* feature of BATS. The splitting points can be observed in Figure. The separated parts are put into *Workspace* as new *DataItems*. The *Multi-Step Procedure* is implemented on the steady-state portion which is named as *Actual.SS* and *Model.SS* respectively.

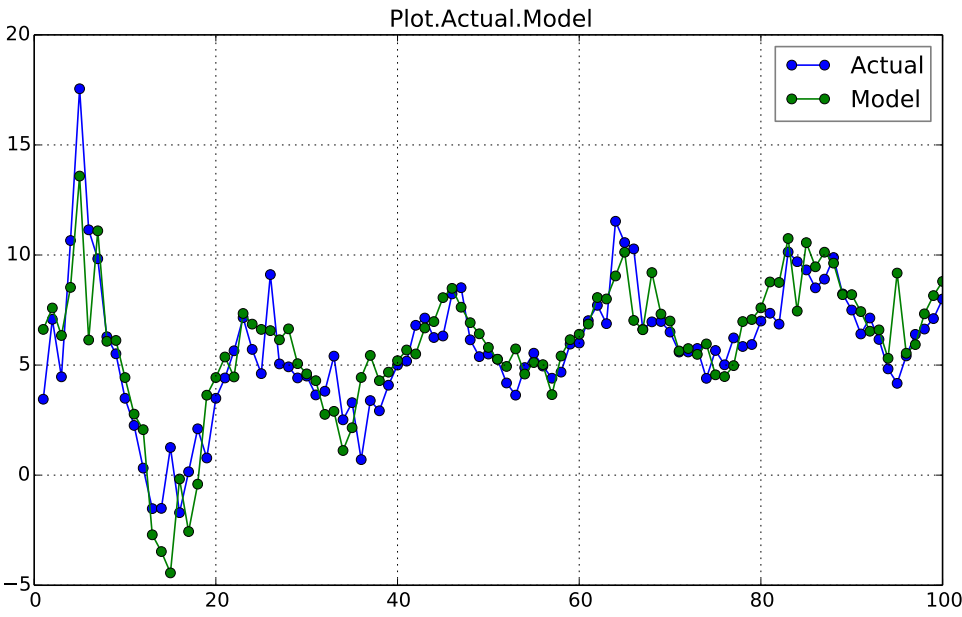


Figure 6.10. Plot for actual and model.

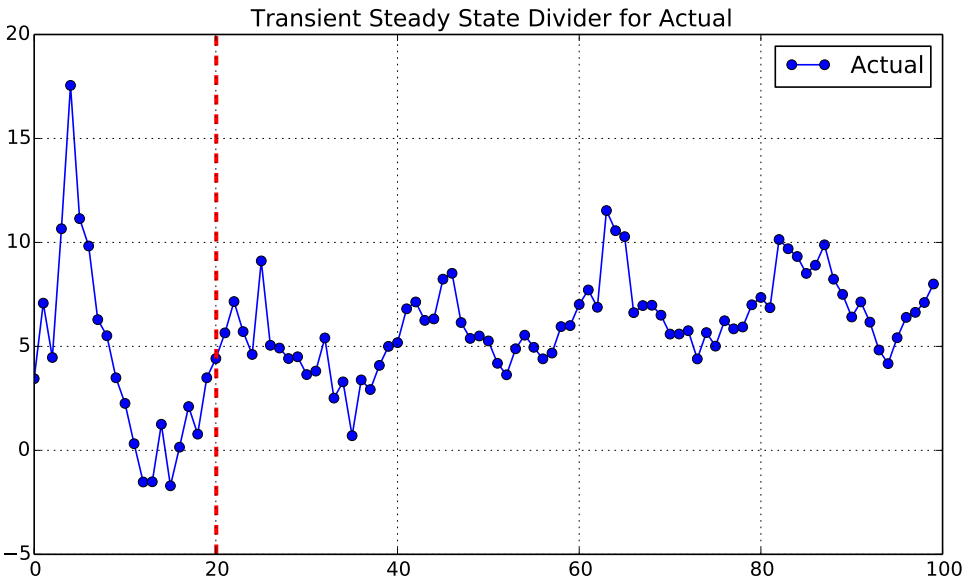


Figure 6.11. Split for actual.

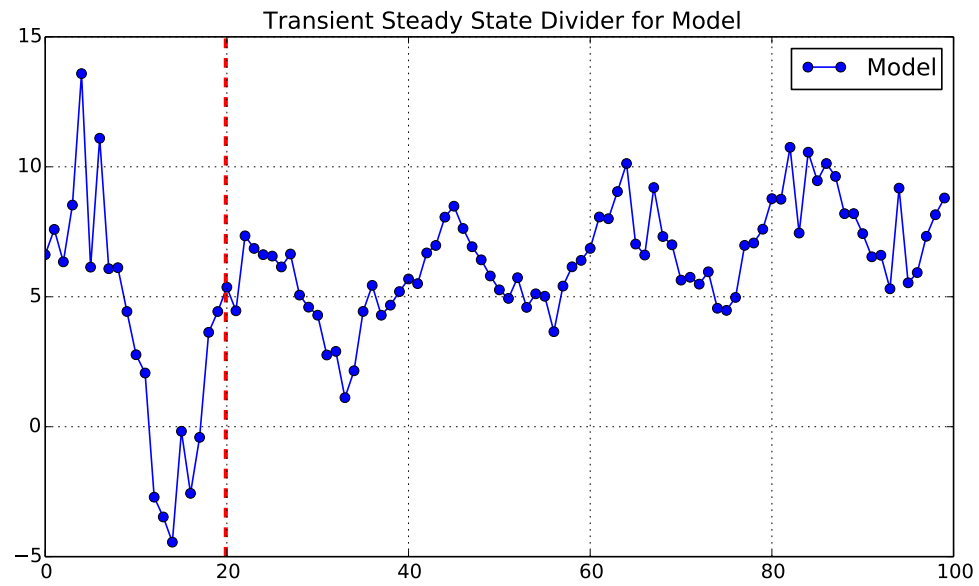


Figure 6.12. Split for model.

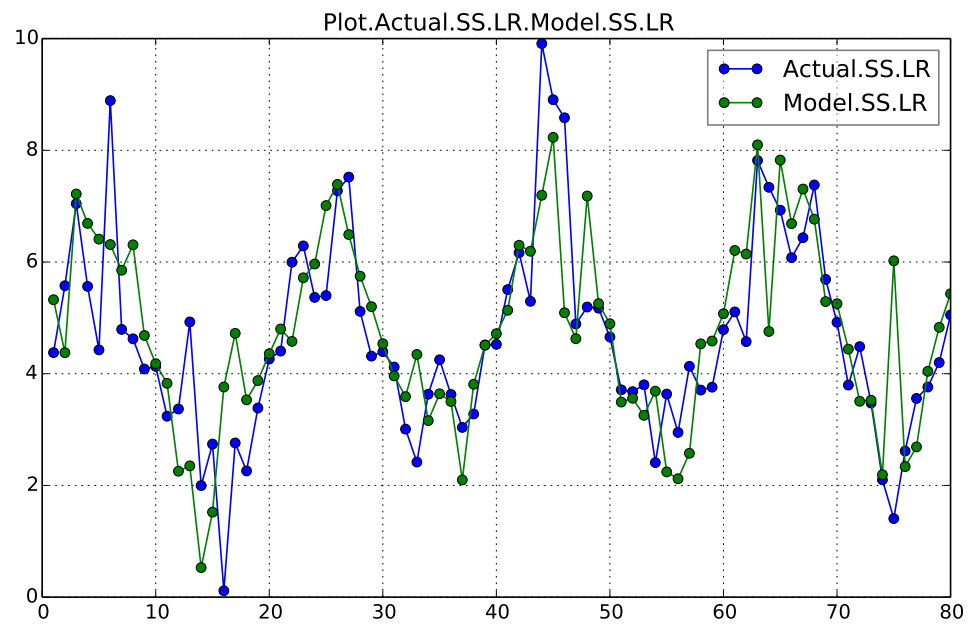


Figure 6.13. Plot for steady-state parts of actual and model after the splitting operation.

Step 1: Trend regression and removal. steady-state parts of the two time-series involve a positive linear trend. This can be clearly observed from the plot in Figure. The first step of the *Multi-Step Procedure* is to remove the trend in data. This application has two implications. Firstly, in order the remaining steps of the procedure, e.g. period estimation using autocorrelation function, to work correctly the time-series should be stationary. Secondly, the slope of the linear trend itself is a metric for behavior pattern similarity. Therefore, it is stored for comparison.

In Figure linear trend regression results are illustrated. From the analysis it is calculated that the slope of linear trend lines are 0 for actual data and 0 for model output. Trend removal operation results in new *DataItems* named as *Actual.SS.LR* and *Model.SS.LR* respectively.

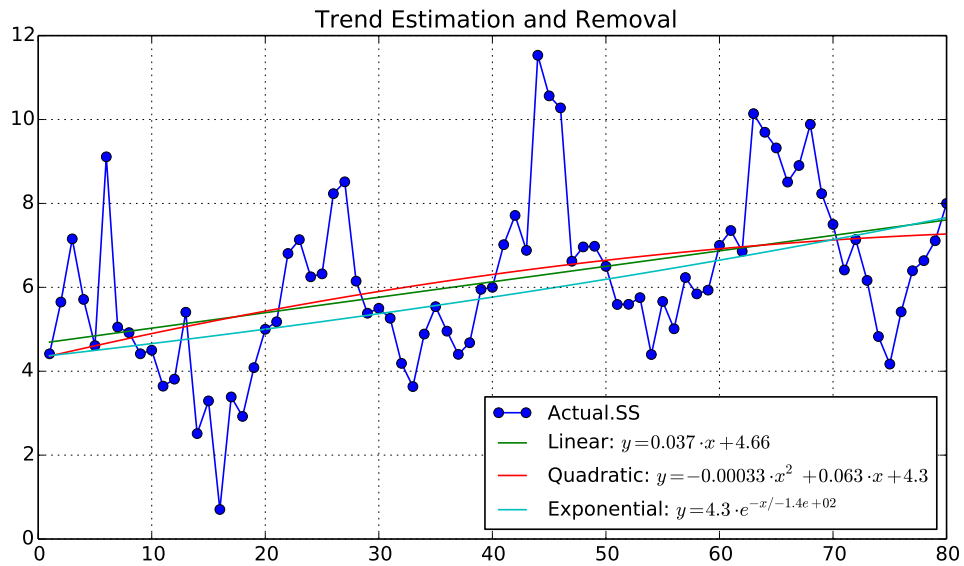


Figure 6.14. Trend regression results for actual.

Step 2: Period estimation. Periods automatically calculated using the period finding algorithm (see Figure 4.5). This algorithm work correctly for periodic behaviors where noise is not very high, and there exists no significant secondary periodicities. For manually estimating the period of a time-series, two methods are implemented in BATS; *Autocorrelation* and *Spectral Densities*. For illustrative purposes the results of these two methods for the actual data and model output are shown in Figure. The

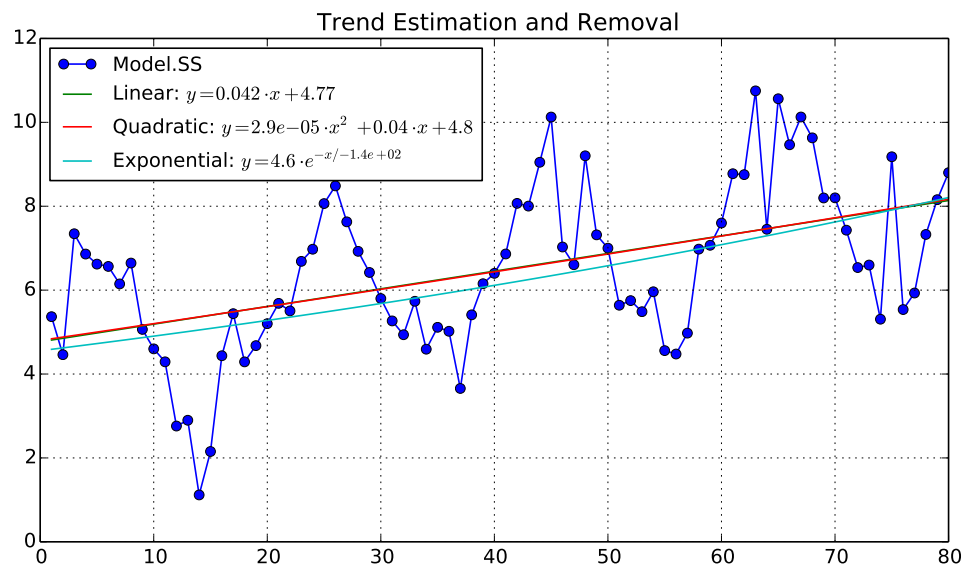


Figure 6.15. Trend regression results for model.

visual inspection of results agree with the automated peak finding algorithm, which finds the periods for both real and simulated as 20.

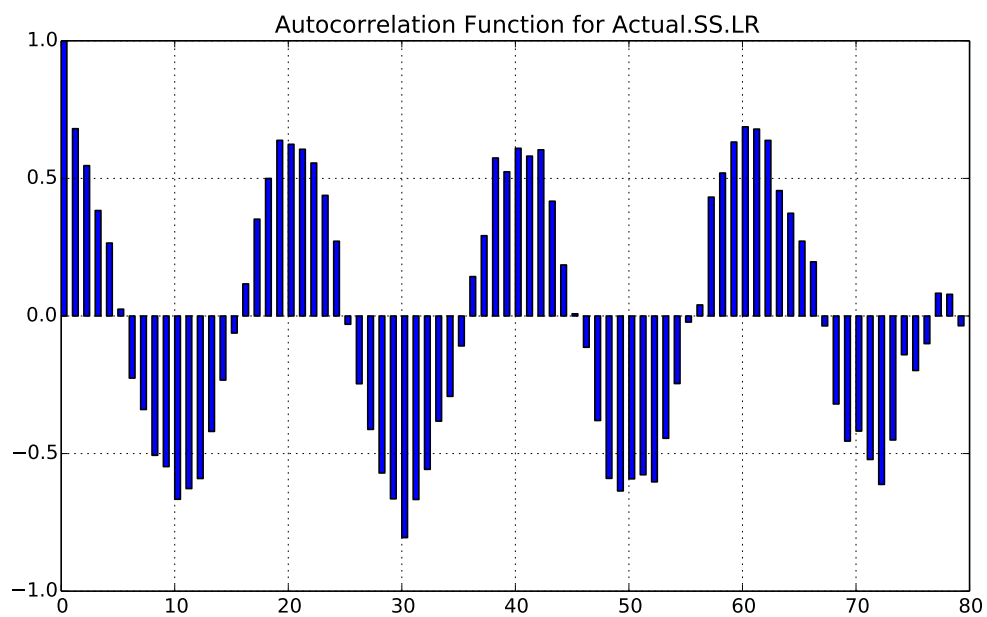


Figure 6.16. Autocorrelation function results for actual.

Step 3: Autocorrelation test. Autocorrelation of two time-series at lags from 0 to

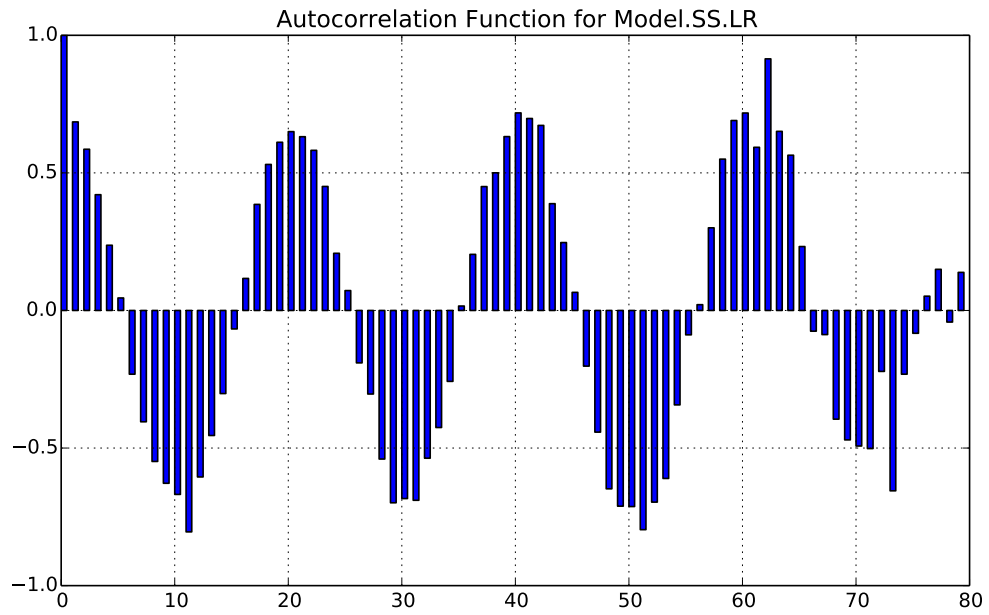


Figure 6.17. Autocorrelation function results for model.

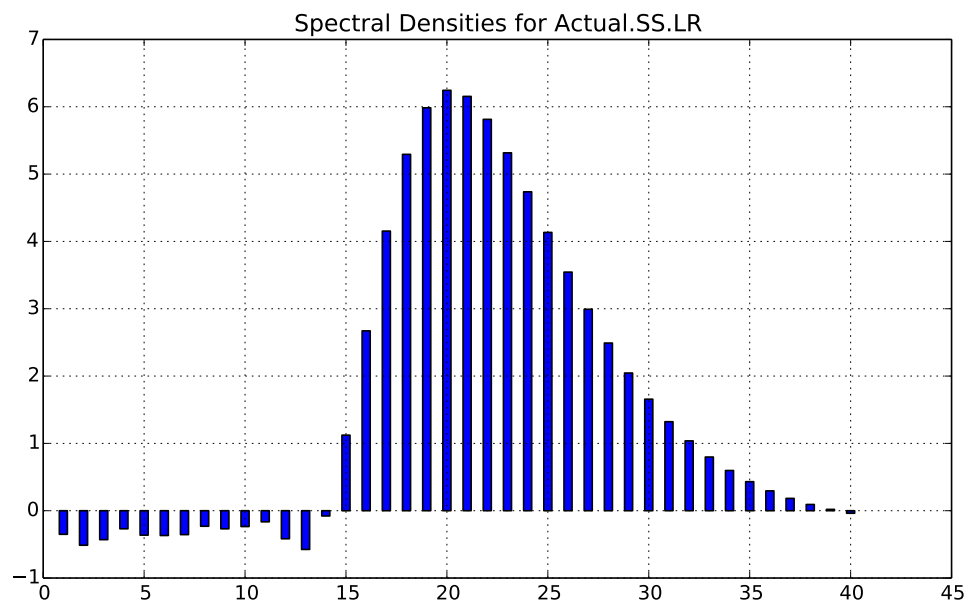


Figure 6.18. Spectral densities of actual.

k can be statistically compared using the *Autocorrelation Test* feature of BATS. This test calculates and constructs a confidence bound on the differences between lags and

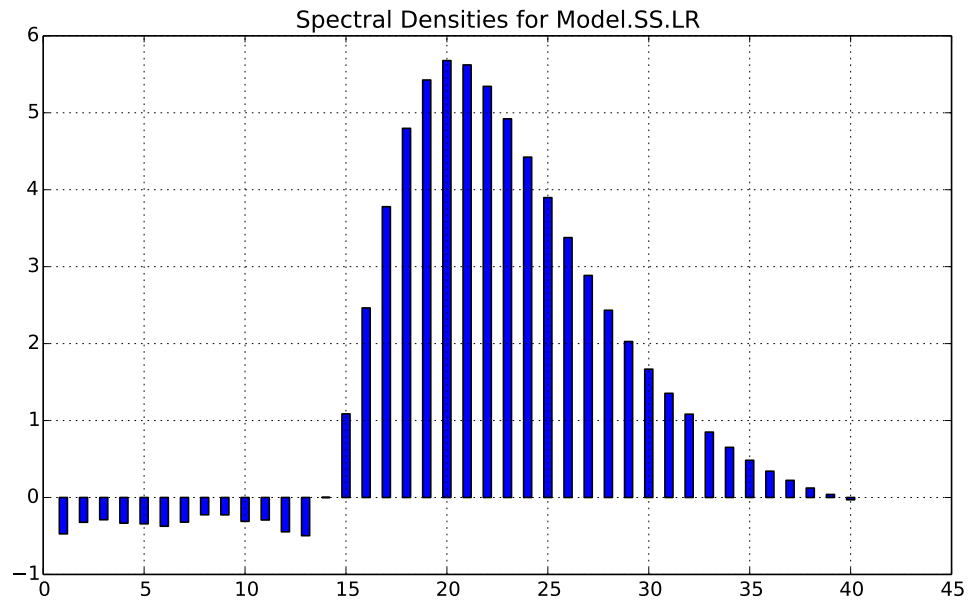


Figure 6.19. Spectral densities of model.

tests if the differences are equal to 0. The actual data and model output passes this test, since the differences between lags lie in the confidence band. This is illustrated in Figure.

Step 4: Compare means and *Step 5: Compare variances*. These statistical metrics are calculated using *Statistics Summary* feature of BATS.

Step 6: Amplitude estimation. The steady-state behavior of the two data series can be categorized as oscillations with constant amplitudes. For this type of behaviors amplitude can be estimated using trigonometric function fitting, which is implemented in the *Amplitude Estimation Constant* feature of BATS. In Figure the results of the method are shown. The estimated amplitude values are 0 for actual data and 0 for model output.

Step 7: Phase lag comparison. Phase lag between two time-series can be computed using *Crosscorrelation* feature of BATS. The results indicate that the two data series are in phase, which can be seen in Figure.

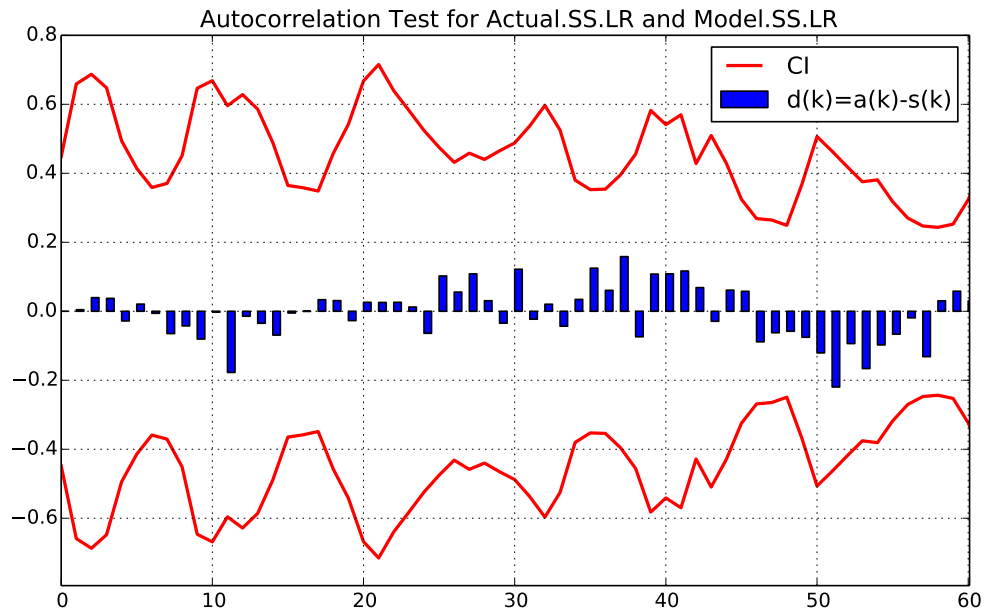


Figure 6.20. Autocorrelation test results for actual and model.

Comparative Stats				
	Actual.SS.LR	Model.SS.LR	Difference	Error:
Mean :	4.657902	4.768117	-0.110216	-0.023662
Var :	3.111435	2.670452	0.440983	0.141730
Single Stats				
U :	0.379063			
MSE :	1.671315			
TheilM :	0.007268			
TheilS :	0.010077			
TheilC :	0.982655			

Figure 6.21. Statistics summary results for actual and model.

Step 8: Coefficients of similarity. *Statistics Summary* feature that is used in steps 3 and 4, calculates and reports coefficients of similarity in addition to mean and variance. These metrics are *discrepancy coefficient* suggested by Barlas (1989) and *theil statistics* suggested by Sterman (1984). Discrepancy coefficient is allowed to take large values if the previous steps of the procedure are passed successfully. It is 0.3790 in this case and as mentioned by Barlas (1996) rather large values are acceptable.

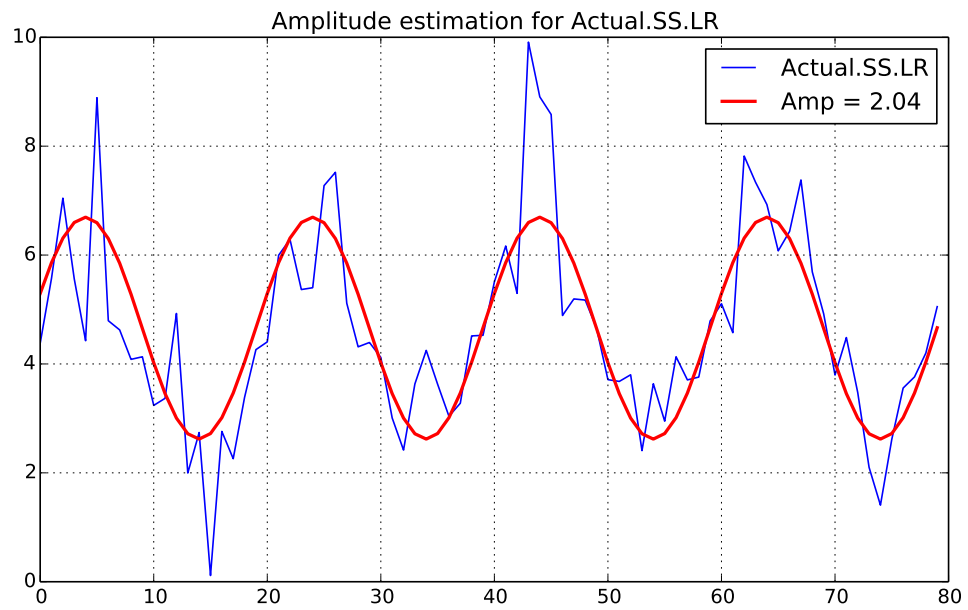


Figure 6.22. Amplitude estimation results for actual.

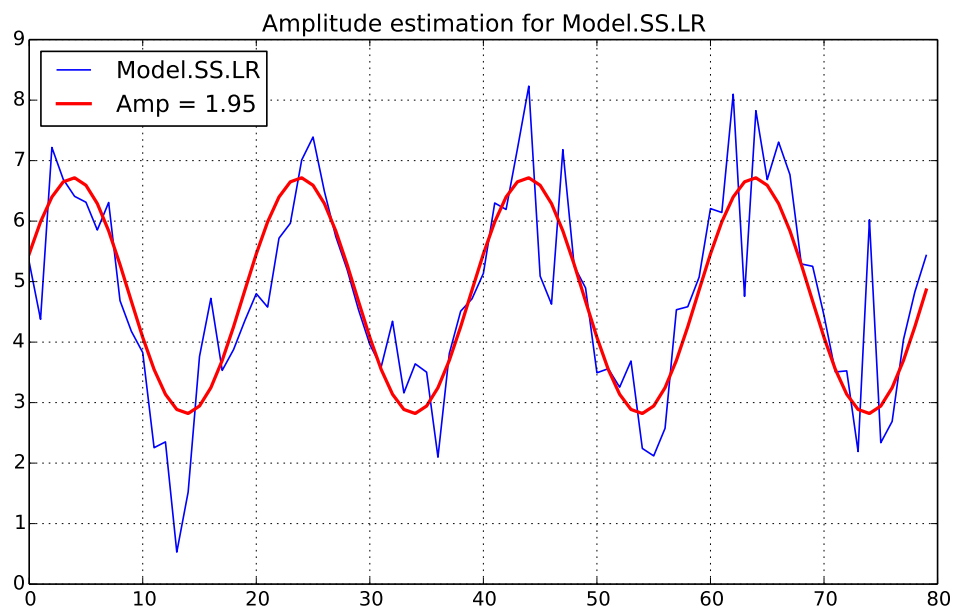


Figure 6.23. Amplitude estimation results for model.

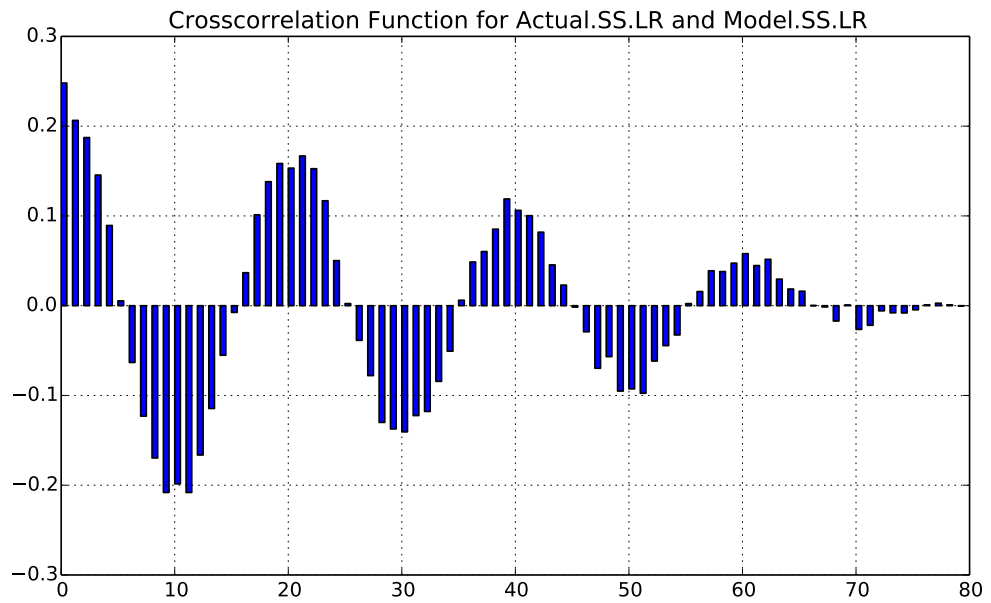


Figure 6.24. Crosscorrelation function results for actual and model.

Step 9: Transient phase analysis. The transient parts of the two data series that are obtained by *Split* tool can also be compared using *Graphical Comparison* feature of BATS. In this demonstration the following metrics are calculated; (i) minimum levels, (ii) maximum levels, (iii) time duration between maximum and minimum observations, and (iv) overall change in stock level. In order to calculate these hypothetical metrics the following options of *Graphical Comparison* tool are used respectively; (i) Y-point comparison, (ii) Y-point comparison, (iii) X-interval comparison, and (iv) Y-interval comparison. The results of this test are illustrated in Figure.

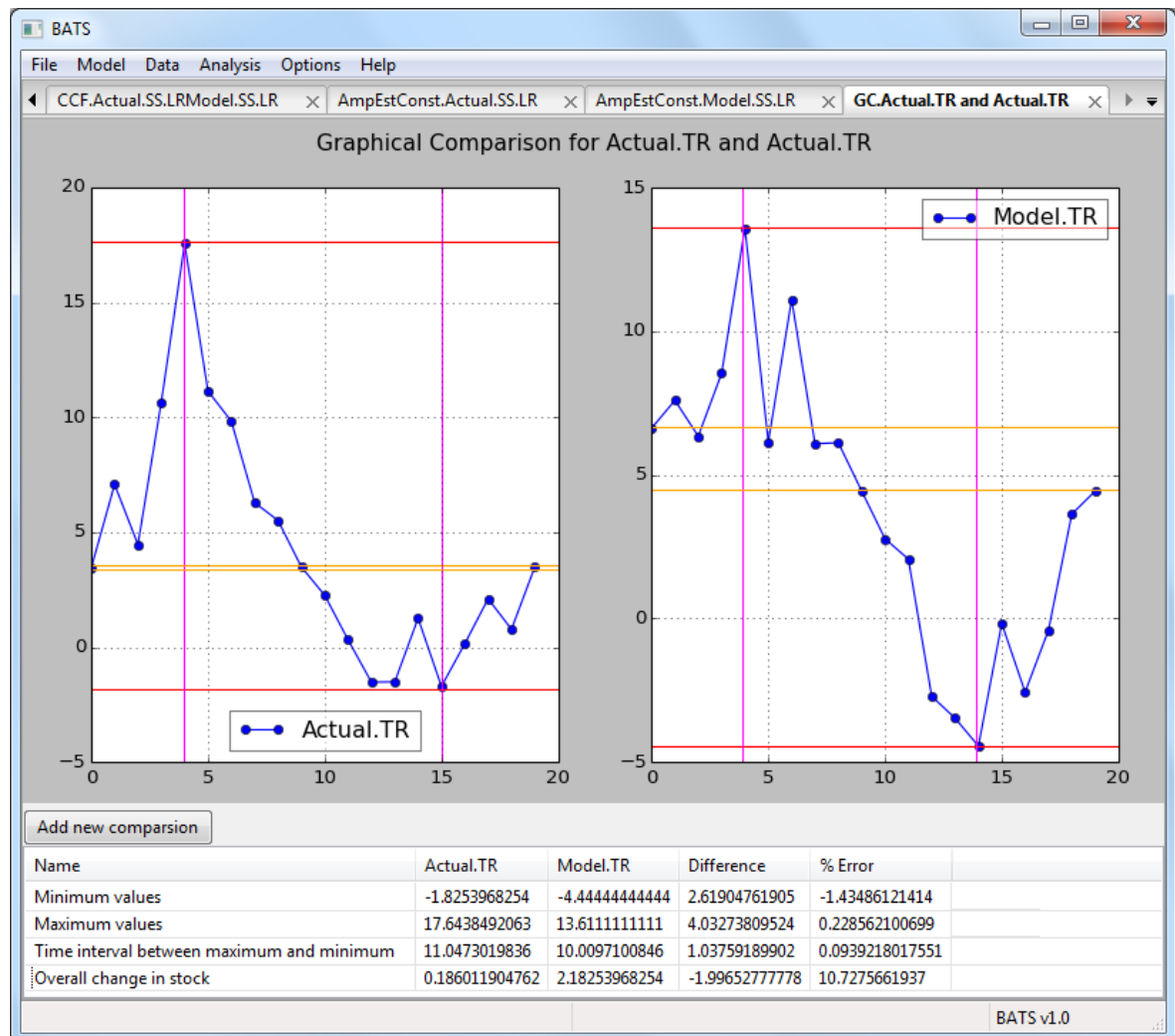


Figure 6.25. Graphical comparison results for actual and model.

Overall results can be seen here in Table 6.1. Conclusion on behavior pattern similarity depends on the allowance of dissimilarity and the importance of point match. Nevertheless, in this example the two data series compared can be argued to pass the

behavior pattern similarity tests.

Table 6.1. Summary table for behavior validity pattern tests.

Metric	Actual	Model	Diff.	% Err.
<i>Steady-state Part</i>				
Linear trend slope	0.0370	0.0420	0.0050	0.1351
Primary period	20	20	0	0.0000
Autocorrelation test	Passed		-	
Mean	4.6579	4.7681	0.1102	0.0237
Variance	3.1114	2.6705	-0.4409	-0.1417
Amplitude	2.0400	1.9500	-0.0900	-0.0441
Phase Lag	0		-	
U	0.3791		-	
MSE	1.6713		-	
UM	0.0073		-	
US	0.0101		-	
UC	0.9826		-	
<i>Transient Part</i>				
Maximum	17.6438	13.6111	-4.0327	-0.2286
Minimum	-1.8254	-4.4444	-2.6190	1.4348
Time between max and min	11.0473	10.0097	-1.0376	-0.0939
Overall change	0.1860	2.1825	1.9965	10.7339

6.3. Sensitivity Analysis

In this demonstration behavior sensitivity analysis of a system dynamics model will be analyzed using *Behavior Space Classifier* feature of BATS. This feature is useful for sensitivity analysis in two aspects. Firstly, the parameter sensitivity on the behavior mode can be analyzed, i.e. “After which parameter values does the model behavior changes its behavior mode from goal-seeking-growth to overshoot-and-decay?”. Second usage mode is to asses the proximity of the model output to specific behavior class, i.e. “Which parameter combination results in higher or lower probability of oscillatory behavior?”.

6.3.1. Test Model: Temperature Adjustment Model

For this demonstration we use the temperature adjustment model. This is a model for the stock control problem under the presence of two types of delays. The first one is the material delay and it represents the delay occurs due to the flow of hot/cold air/water in the air-conditioner system. The second one is the information delay and it represents the perception delay of the human being while feeling and measuring the actual temperature. The stock-flow diagram of the model can be seen in Figure 6.26. For simplicity model variables are represented by symbols and letters. The equations of the model are as follows:

$$\frac{dX(t)}{dt} = \frac{X^* - Y(t)}{\alpha}$$

$$\frac{dY(t)}{dt} = \frac{X(t) - Y(t)}{\beta}$$

6.3.2. Tests and Results

Step 1: Click *Behavior Space Classifier* under *Model* menu. Using the file selection dialog locate the model file *temperature_adjustment.vpm*.

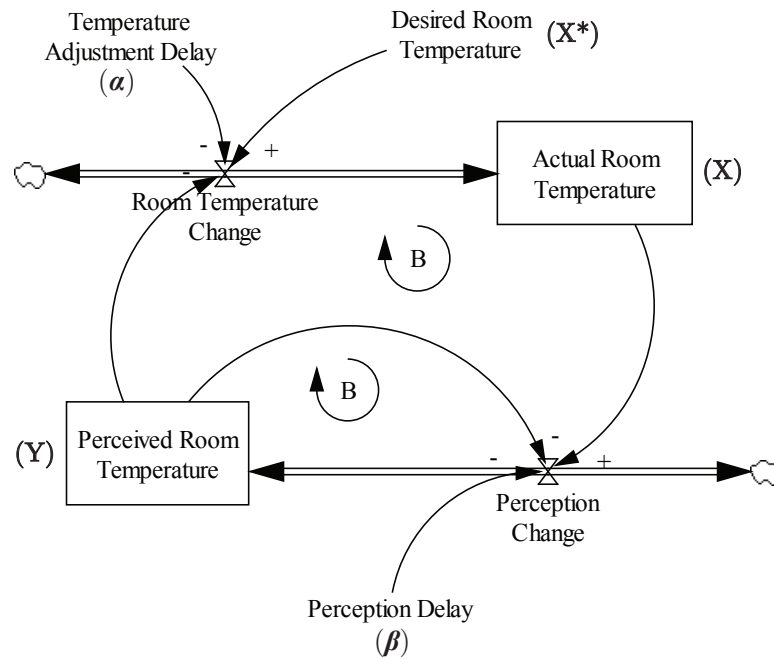


Figure 6.26. Stock-flow diagram of the temperature adjustment model.

Step 2: A custom dialog box appears. The user enters the following information:

- Parameter 1, start, stop, num
- Parameter 2, start, stop, num
- Outcome of interest
- Search type

Figure 6.27. Settings for maximum likelihood class search.

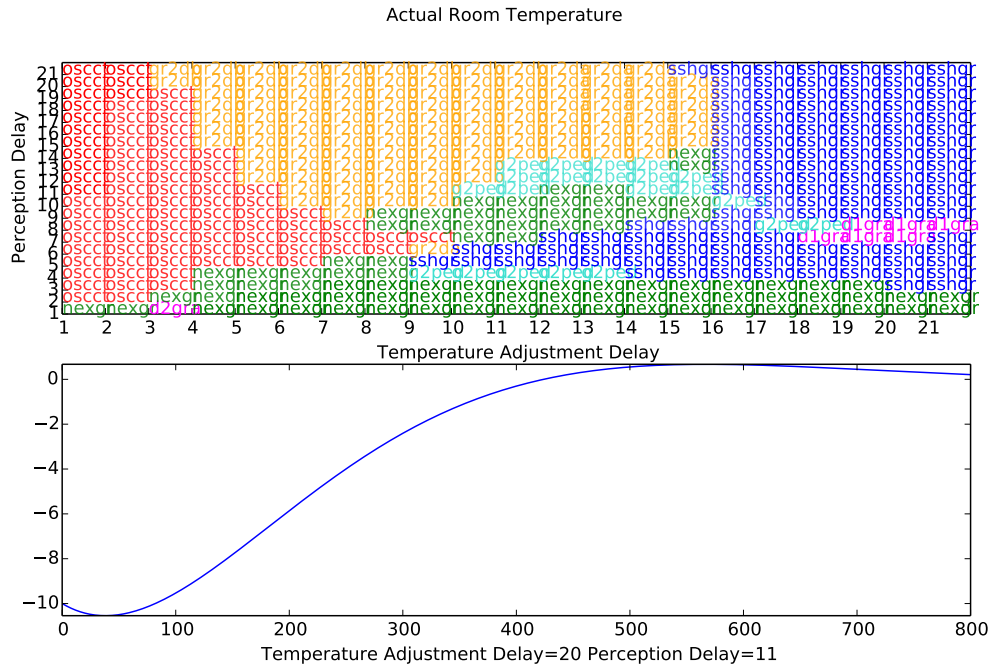
Parameter 1 and 2 are the uncertainty parameters. The user specifies their sensitivity entering start and stop values and number of points. These values are used to

create sensitivity ranges of equal intervals.

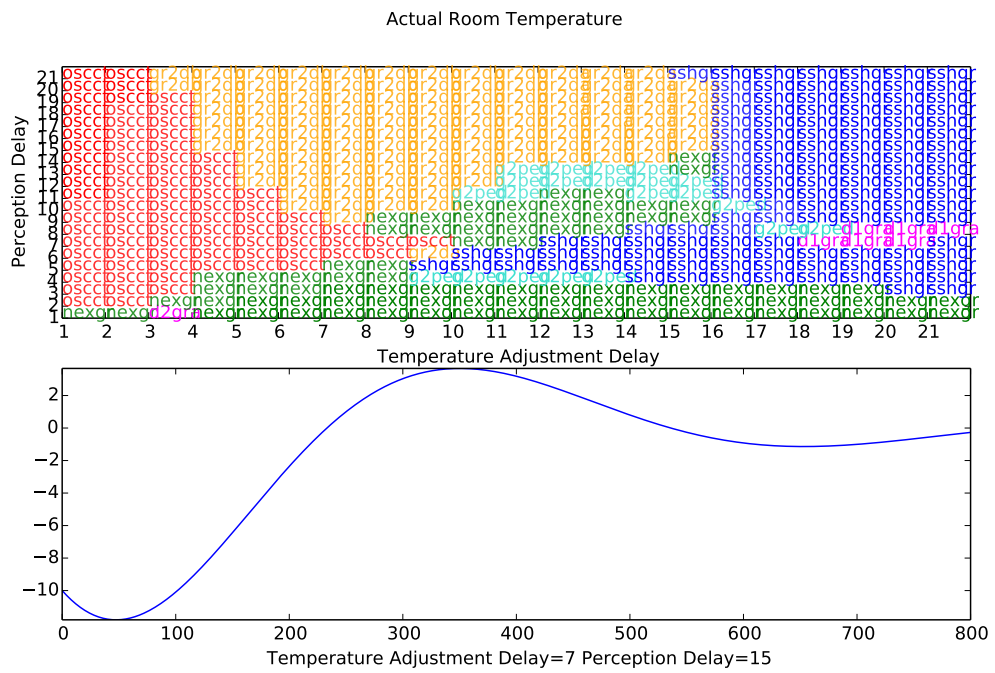
The program searches the feasible region for all possible combinations and stores the individual time-series and the series' pattern class probabilities. So if the user specifies N_1 alternative values for parameter 1 and N_2 for parameter 2, the program performs $N_1 \times N_2$ many simulations.

There are 2 alternatives for search type. In the first option highest class probability is found, stored and displayed in the final graph, this is represented by 'all' option in the selections box. In the second option, the user specifies the s-shaped growth pattern, and specifies 'sshgr' option. In this method the since the program is able to work on real numbers compared to class labels, a contour plot is displayed.

Step 3: Visualizing the results. In the plot with labels the user can navigate between the individual runs by clicking on the corresponding cell on the top plot.

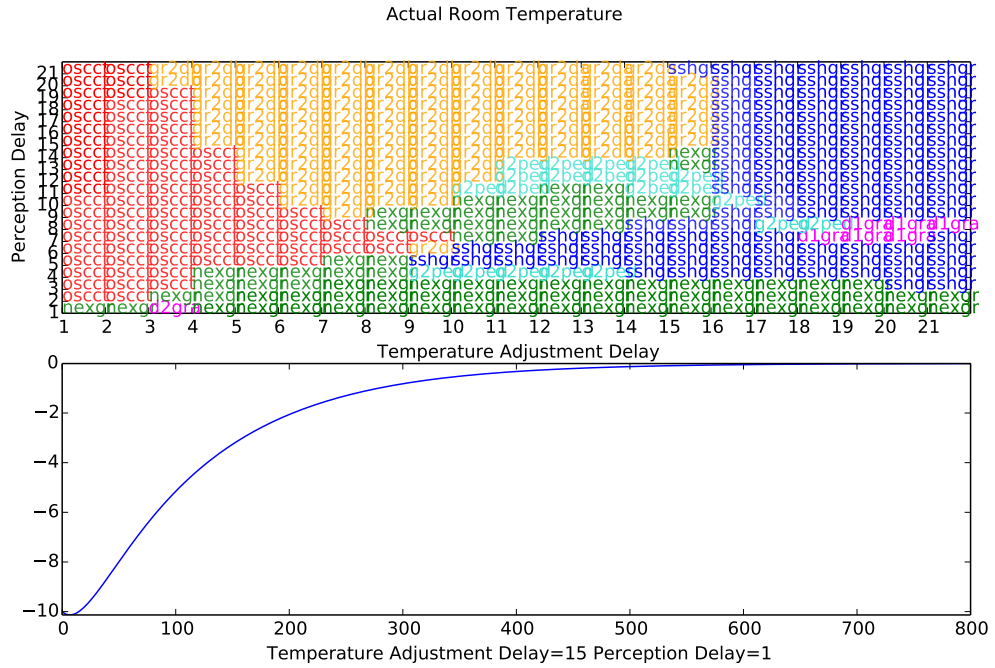


(a) SSHGR

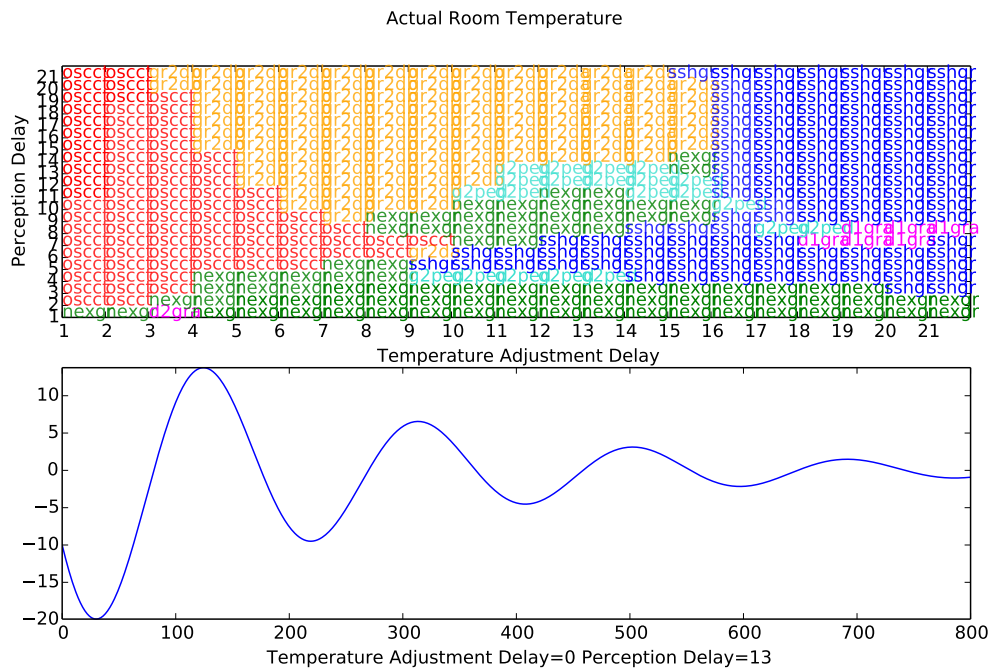


(b) GR2DB

Figure 6.28. *Behavior Space Classifier* results. Different views in bottom plot are obtained by clicking on the top grid.



(a) NEXGR

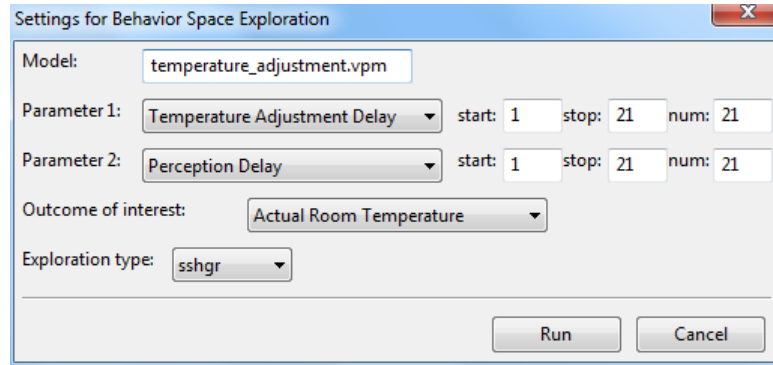


(b) OSCCT

Figure 6.29. *Behavior Space Classifier* results continued.

6.4. Parameter Calibration

In this demonstration we suppose that the desired behavior for the *Actual Room Temperature* variable in the temperature adjustment model, is an S-shaped growth pattern. There are two decision parameters of sensitivity, namely *Temperature Adjustment Delay* and *Perception Delay*. The feasible ranges for these parameter are determined as $[1, 21]$. The modeler is interested in the behavior space of the model with respect to similarity to generic s-shaped growth pattern. In order to perform this task, *Behavior Class Mapper* feature can be used. The setup of this analysis can be seen in Figure 6.30.



Settings for Behavior Space Exploration

Model: temperature_adjustment.vpm

Parameter 1: Temperature Adjustment Delay start: 1 stop: 21 num: 21

Parameter 2: Perception Delay start: 1 stop: 21 num: 21

Outcome of interest: Actual Room Temperature

Exploration type: sshgr

Run Cancel

Figure 6.30. Settings for *sshgr* likelihood search.

The method produces the plot that can be seen in Figure 6.31. By observing this figure, the following conclusion can be made; in order to calibrate the model behavior to an s-shaped growth pattern the best parameter combination is either $\alpha = 12.5$ and $\beta = 5$, or $\alpha = 20$ and $\beta = 3$, where α denotes *Temperature Adjustment Delay* and β denotes *Perception Delay*.

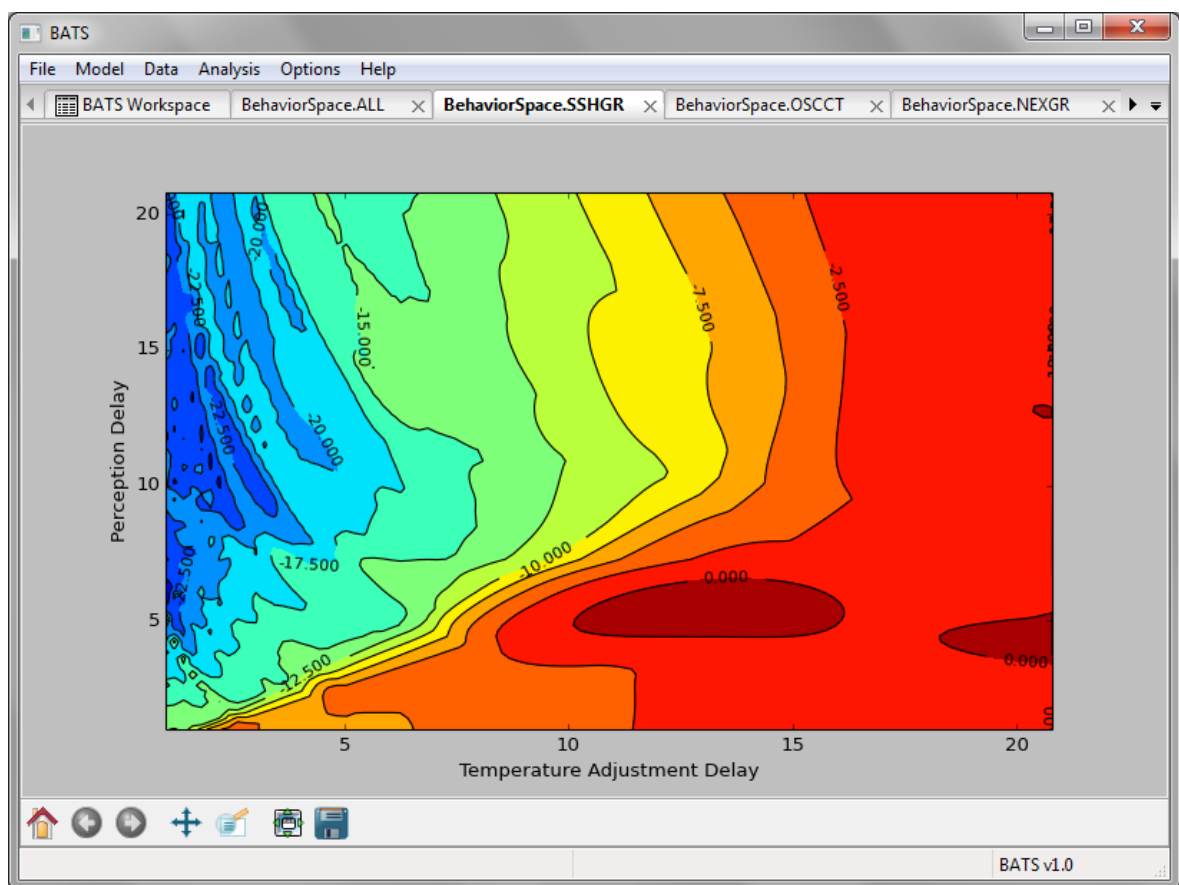


Figure 6.31. Results for *sshgr* likelihood search.

6.5. Policy Design

In this demonstration we suppose that the objective of the analyst is to find a policy that avoids overshooting and oscillating behavior. In this demonstration temperature adjustment model is used. Policy variable is determined as *Actual Room Temperature*. There are two decision parameters related with policy design *Temperature Adjustment Delay* and *Perception Delay*. The feasible ranges for these parameter are set as $[1, 21]$, similarly to in the previous demonstrations. In order to perform this task, *Behavior Class Mapper* feature can be used. The setup of this analysis can be seen in Figure 6.32. In order to avoid overshooting and oscillating behavior, goal seeking growth (*nexgr*) is specified as the behavior class to be assessed.

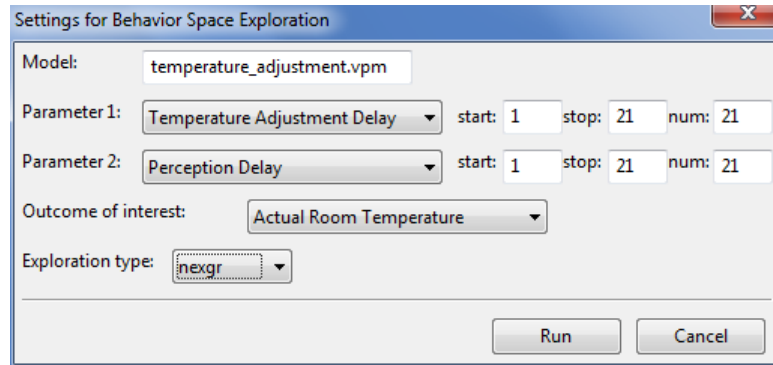


Figure 6.32. Settings for *nexgr* likelihood search.

The results from *Behavior Class Mapper* method suggests that the goal seeking behavior is most likely to be observed when *perception Delay* is as small as possible and *Temperature Adjustment Delay* is around 14. See Figure 6.33 for the resulting screen. So as long as the analyst selects the parameter values in the area indicated by *dark red* color, the policy objective would be satisfied.

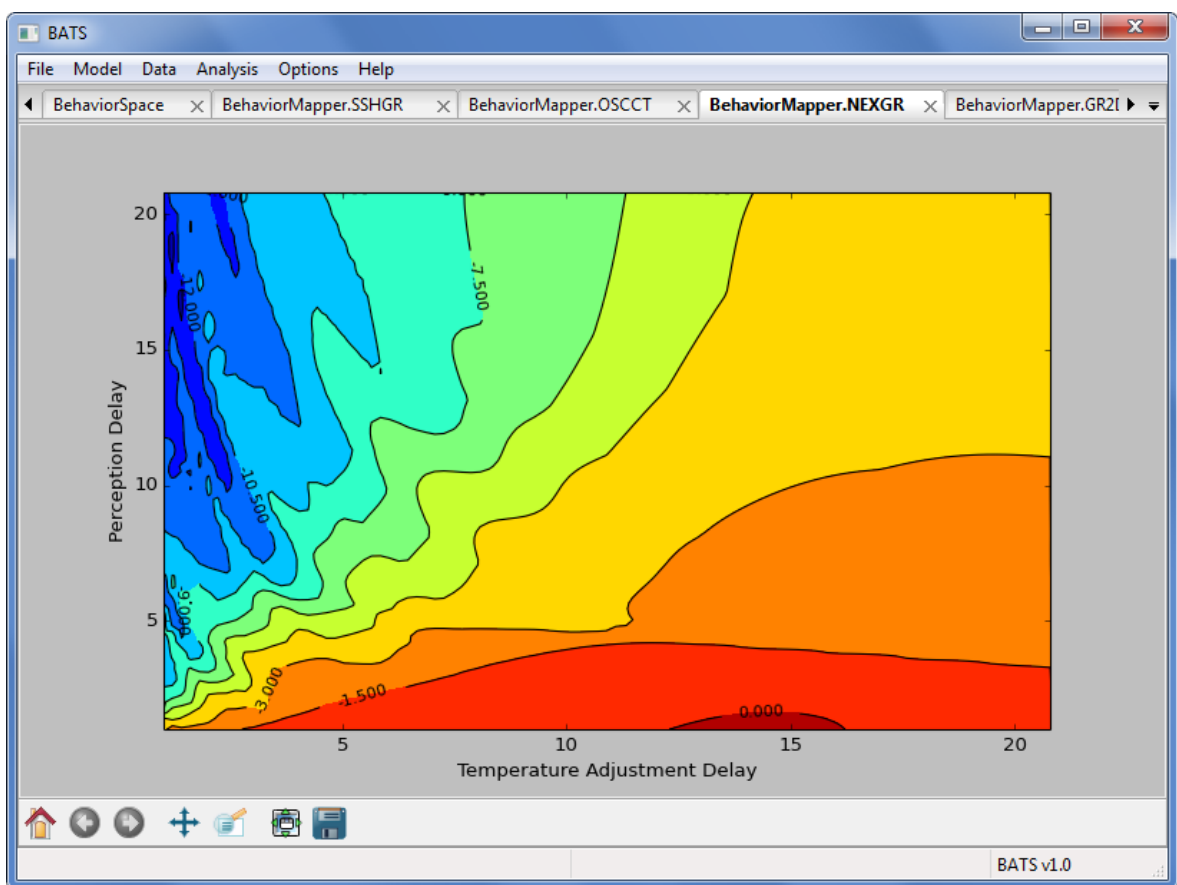


Figure 6.33. Results for *nexgr* likelihood search.

6.6. Policy Analysis

BATS can be used for policy analysis purposes. In this example a simulation model developed on Vensim is connected to the program. This connection is accomplished via *Model Docking Window* feature of BATS. Through this feature the user can set different parameter values and observe results of outcome variables of choice. In addition to visual display of results, the user can export the results to *Workspace* in order to make more detailed analysis on the model output.

6.6.1. Test Model: Predator-Prey Model

This is a simplistic model for two-species population model, in which there are predators and preys. It is assumed that growth of predators is dependent on the availability of preys, and decrease of preys is dependent on the availability of predators. In Figure 6.34 the stock-flow diagram of the model is presented. From the diagram it can be observed that the model structure is customized for this policy problem by introducing one new outflows for each stock, namely *harvestsX* and *harvestsY*. These outflows are necessary for making decisions on the harvesting efforts. Hence, the policy parameters become hx and hy respectively. As hx increases the proportional harvesting of X stock increases.

Mathematical equations of the model can be given as:

$$\frac{\partial X}{\partial t} = \alpha X - \beta XY - h_x X$$

$$\frac{\partial Y}{\partial t} = -\gamma Y + \delta XY - h_y Y$$

6.6.2. Policies and Results

Suppose that a governmental institute is currently implementing a hunting policy and wants to evaluate three new alternative policies. There are harvesting/hunting

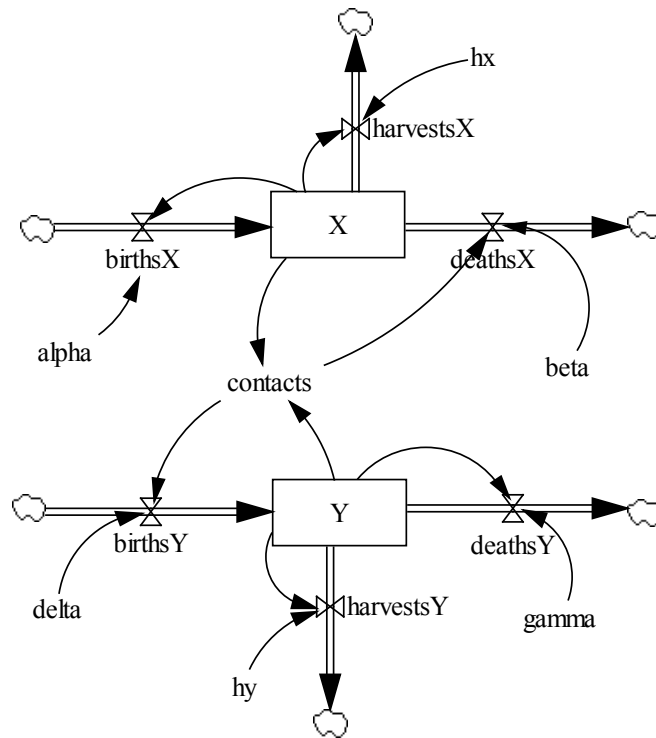


Figure 6.34. Stock-flow diagram of the predator-prey model.

activities both predators and preys, and the new policies are based on increasing the rate of these activities. Three alternative policies are increase harvesting of preys, increase harvesting of predators and increase harvesting of both. As a result, there are four policies, which can be observed in Table 6.2. Also in this table, parametric representations of the policies are given. The institution is doubling the proportional harvesting fractions; h_x denotes the harvest fraction of preys and h_y denotes the harvest fraction of predators. Policy name “harvest preys” stands for doubling the harvest fraction of preys.

Table 6.2. Parametric representations of policy alternatives regarding the predator-prey model.

Policies	h_x	h_y
Base run: Do-nothing	0.05	0.10
Policy 1: Harvest preys	0.10	0.10
Policy 2: Harvest predators	0.05	0.20
Policy 3: Harvest both	0.10	0.20

Suppose that policy objective is based on the population of preys. The institution is interested in the following metrics about the prey population; equilibrium level, amplitude of fluctuations, minimum level and maximum level. Depending on the policy objective desired behavior pattern characteristics changes, therefore the optimum policy to be selected changes. Firstly, all the relevant information is to be collected. For this case study demonstration, the simulation model developed on Vensim is used. In the following pages the steps taken for preparation of the model outputs are illustrated.

Step 1: Connect the model to BATS using Model Docking Window.

Step 2: Add simulation runs using the Run Specifications dialog box.

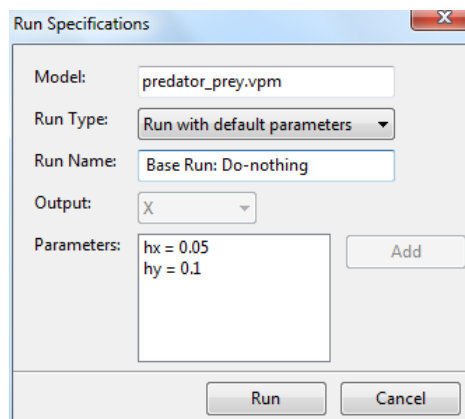


Figure 6.35. Run specifications for base policy.

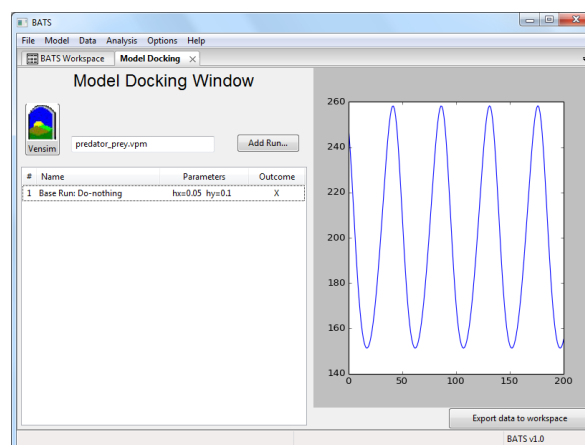
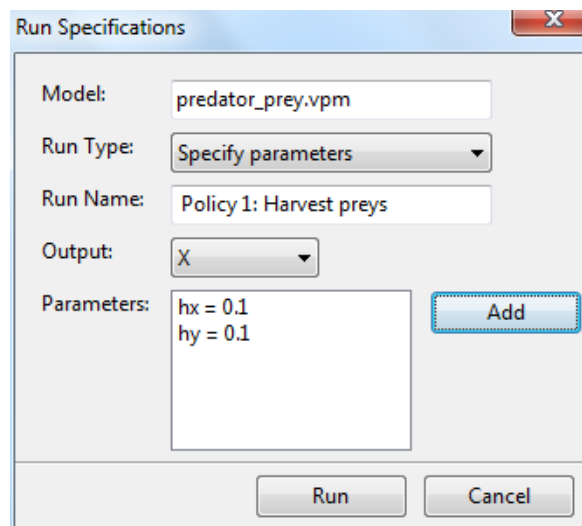


Figure 6.36. Simulation results for base policy.

Step 3: Repeat Step 2 for the remaining three policy alternatives.



The 'Run Specifications' dialog box is shown. It contains the following fields and controls:

- Model:** predator_prey.vpm
- Run Type:** Specify parameters (dropdown menu)
- Run Name:** Policy 1: Harvest preys
- Output:** X (dropdown menu)
- Parameters:**
 - hx = 0.1
 - hy = 0.1
- Add:** button
- Run:** button
- Cancel:** button

Figure 6.37. Run specifications for policy 1.

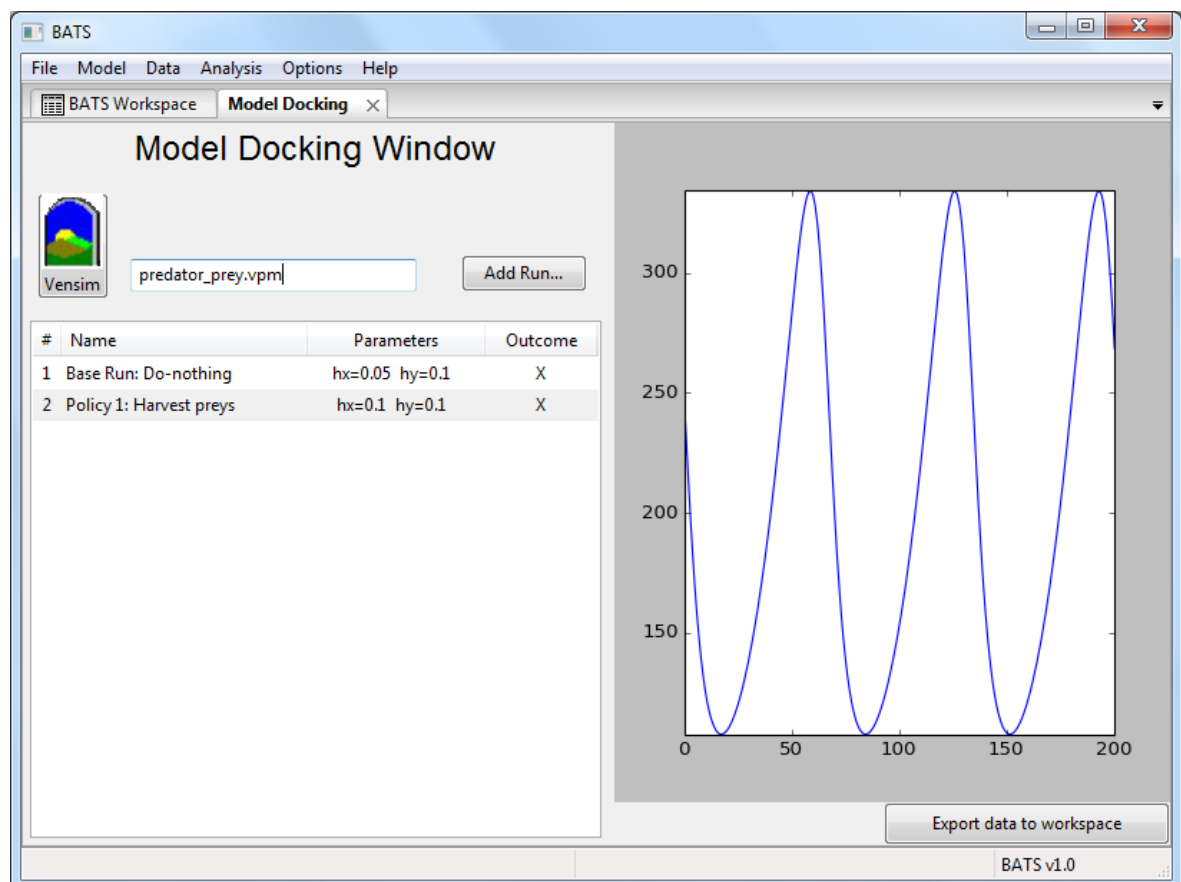
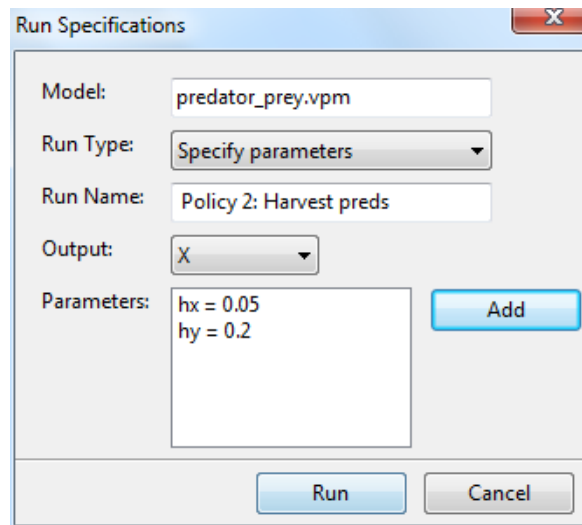


Figure 6.38. Simulation results for policy 1.



Run Specifications

Model: predator_prej.vpm

Run Type: Specify parameters

Run Name: Policy 2: Harvest preds

Output: X

Parameters: hx = 0.05
hy = 0.2

Add

Run Cancel

Figure 6.39. Run specifications for policy 2.

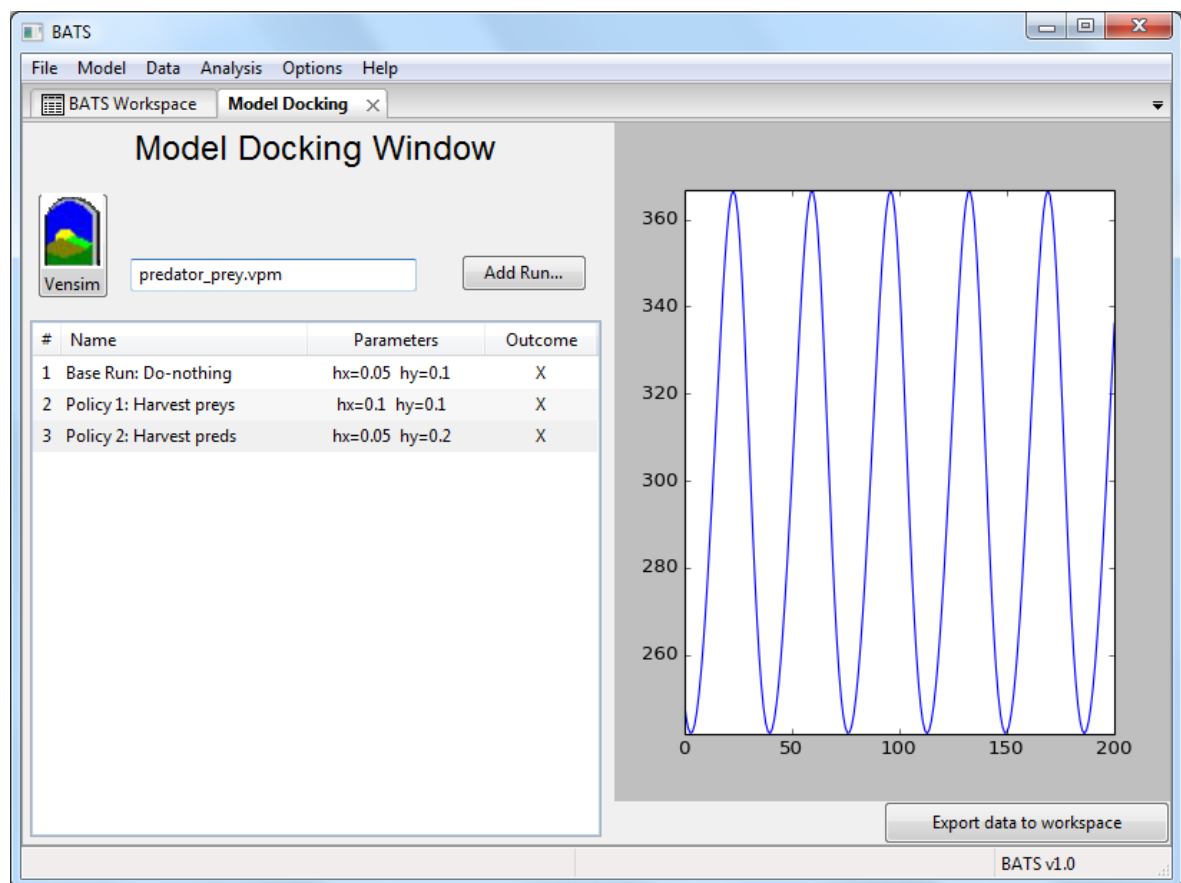
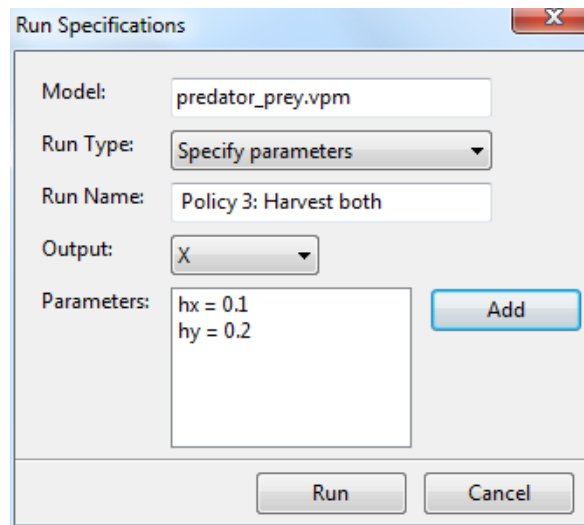


Figure 6.40. Simulation results for policy 2.



Run Specifications

Model: predator_preym.vpm

Run Type: Specify parameters

Run Name: Policy 3: Harvest both

Output: X

Parameters: hx = 0.1
hy = 0.2

Add

Run Cancel

Figure 6.41. Run specifications for policy 3.

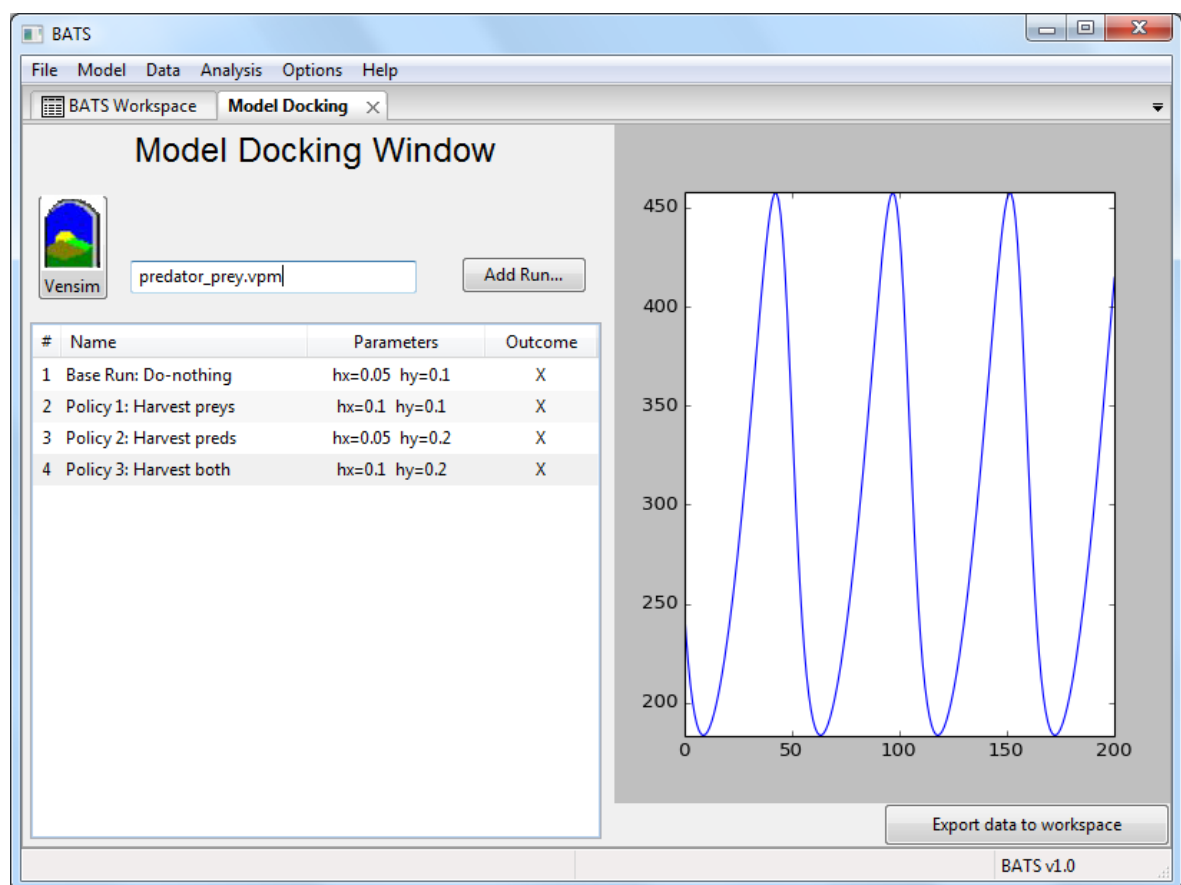
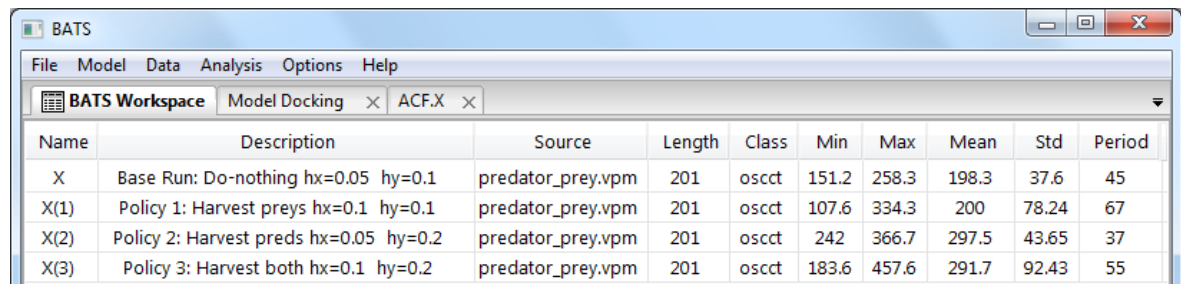


Figure 6.42. Simulation results for policy 3.

Step 4: Export model outputs from the *Model Docking Window* to *Workspace* in order to obtain policy metrics equilibrium level, amplitude, minimum and maximum values.



The screenshot shows the BATS software interface. The 'BATS Workspace' window is active, displaying a table of simulation results. The table has columns for Name, Description, Source, Length, Class, Min, Max, Mean, Std, and Period. There are four rows of data, each representing a different policy scenario.

Name	Description	Source	Length	Class	Min	Max	Mean	Std	Period
X	Base Run: Do-nothing $hx=0.05$ $hy=0.1$	predator_preym.vpm	201	oscct	151.2	258.3	198.3	37.6	45
X(1)	Policy 1: Harvest preys $hx=0.1$ $hy=0.1$	predator_preym.vpm	201	oscct	107.6	334.3	200	78.24	67
X(2)	Policy 2: Harvest preys $hx=0.05$ $hy=0.2$	predator_preym.vpm	201	oscct	242	366.7	297.5	43.65	37
X(3)	Policy 3: Harvest both $hx=0.1$ $hy=0.2$	predator_preym.vpm	201	oscct	183.6	457.6	291.7	92.43	55

Figure 6.43. Workspace after exporting all of the simulation results.

Step 5: Estimate amplitudes using the *Amplitude Estimation Constant* feature. Note that *Workspace* provides quick information on average, minimum and maximum values of the data series. The only pattern characteristics of interest left for this policy analysis, is the amplitude of fluctuations, which can be found by trigonometric function fitting method. This feature also enables the user to try different trigonometric function fitting parameters using sliders for amplitude, phase and bias.

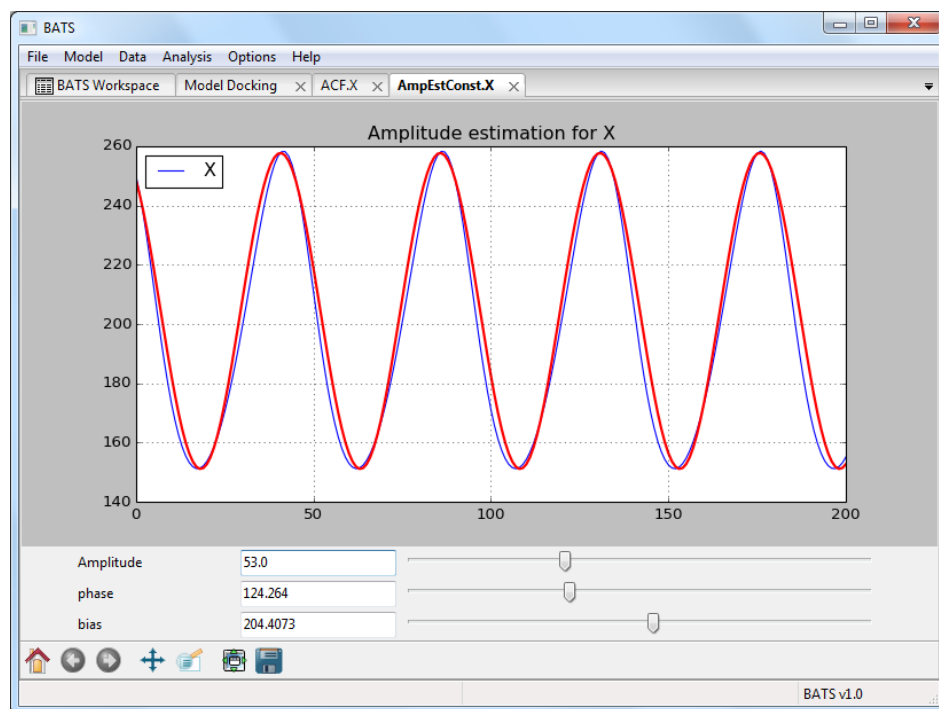


Figure 6.44. Amplitude estimation for base policy, ($Amp. = 53$).

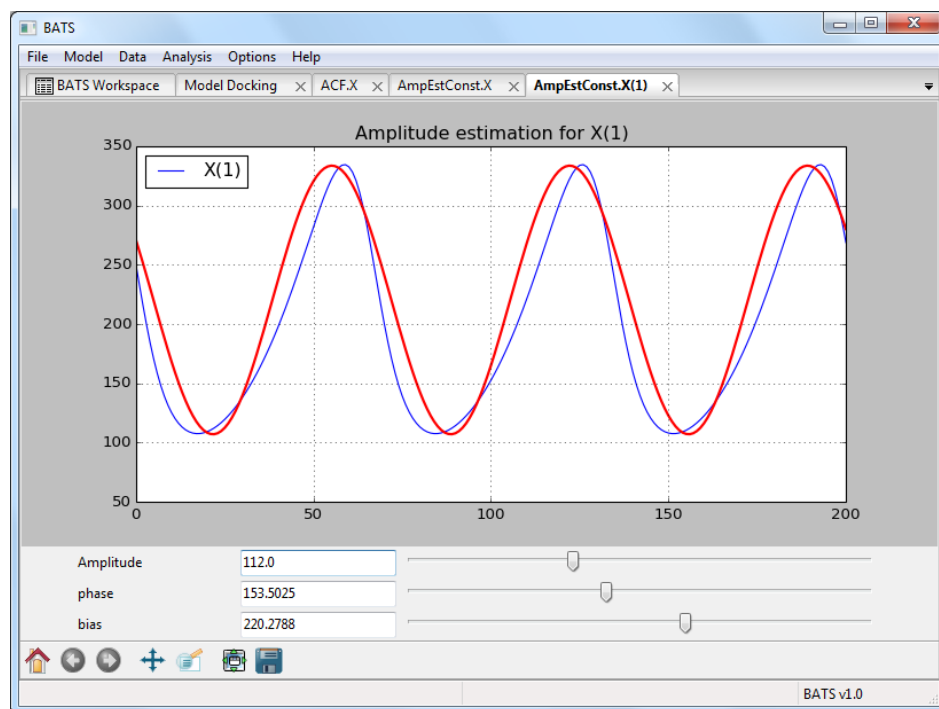


Figure 6.45. Amplitude estimation for base policy, ($Amp. = 112$).

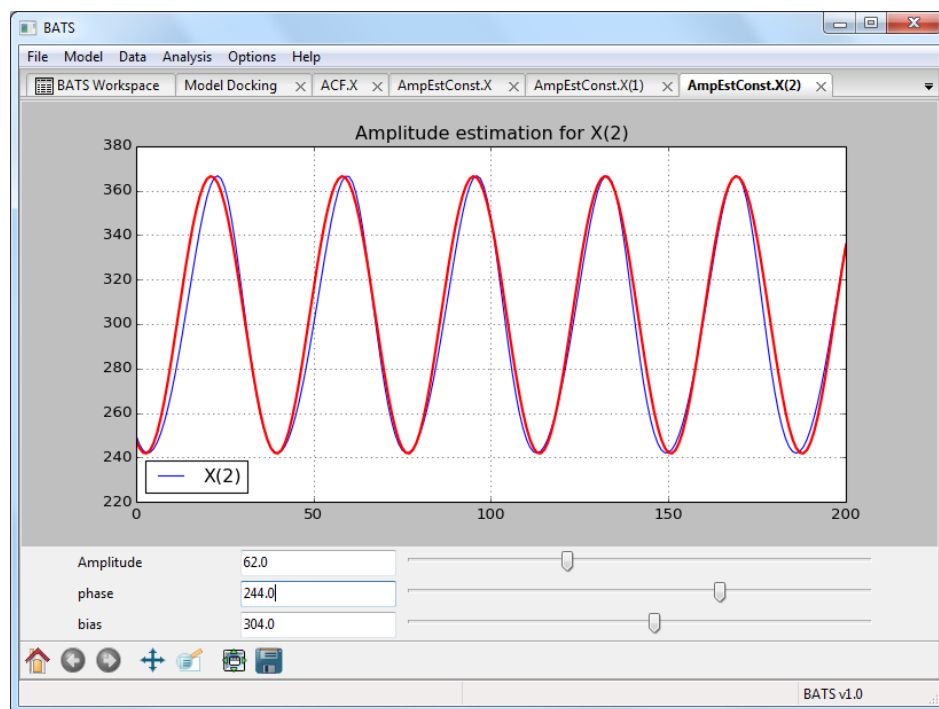


Figure 6.46. Amplitude estimation for base policy, ($Amp. = 62$).

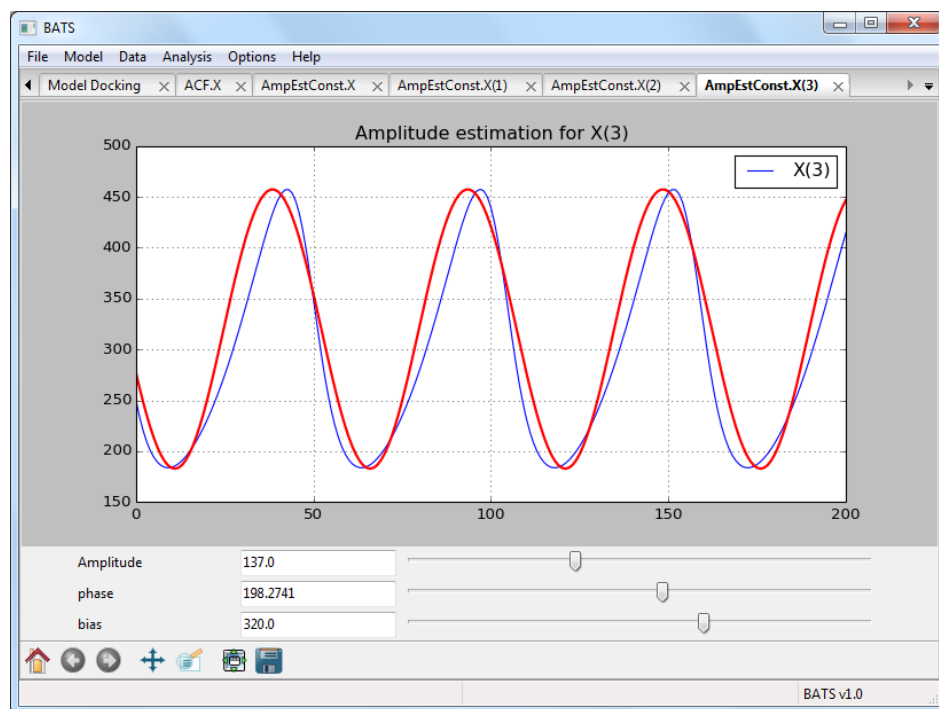


Figure 6.47. Amplitude estimation for base policy, ($Amp. = 137$).

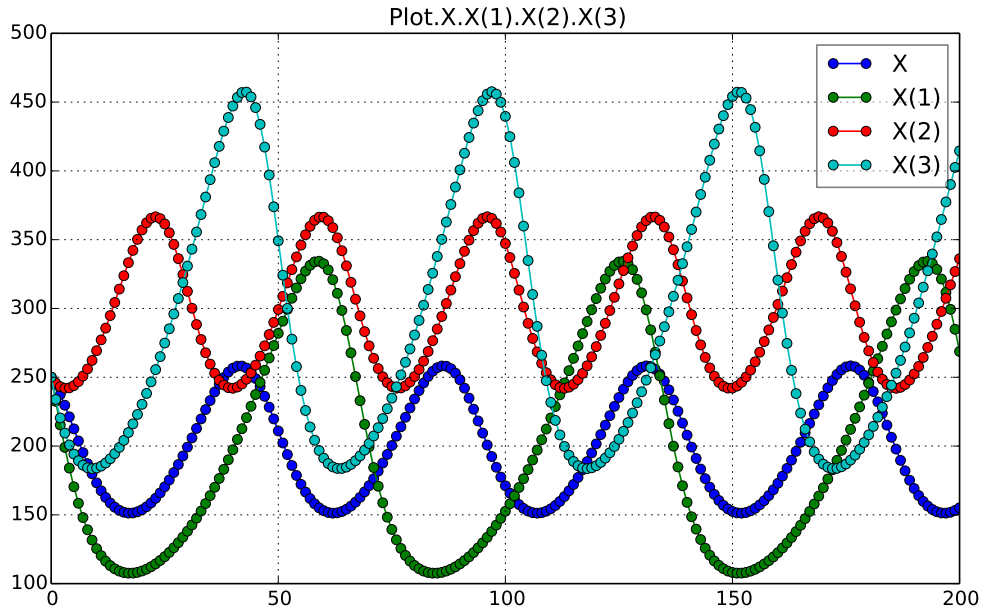


Figure 6.48. Behavior of X (prey population) for all policies.

Table 6.3 summarizes the policy metrics for the four policy alternatives illustrated in Figure 6.48 in this case study. Using these metrics, policy implementations can be achieved. For example; if the policy objective is to increase mean value of prey population then Policy 2 becomes optimal, whereas if the government defines the policy objective as minimize the oscillations, then base policy is favorable. In case of a multi-criteria decision making such as increase overall number of population while maintaining a “low” level of fluctuations, then Policy 2 again becomes the optimal policy.

Table 6.3. Analysis of model outputs for the policy alternatives.

Policies	Eq. (Mean)	Amp.	Min	Max
Base run	198.3	53	151.2	258.3
Policy 1	200.0	112	107.6	334.3
Policy 2	297.5	62	242.0	336.7
Policy 3	291.7	137	183.6	457.6

6.7. Summary of Demonstration and Usage Modes Chapter

In this chapter analysis methods that are offered by BATS are demonstrated for various purposes. These examples span most of the individual features presented and described in Chapter 5. Summary information is provided in Table 6.4. Note that certain features such as *Behavior Class Mapper* and *Amplitude Estimation - Constant* are used for different tasks. This is applicable to other features depending on the purpose of the study.

Table 6.4. Summary of features used in this chapter.

Task	Model/Data	Features Used
Extreme Condition Testing	density dependent growth model	Hypothesis Tester
Behavior Pattern Validity Testing	oscillatory data from spreadsheet and text file	Load Data from File Split Trend Autocorrelation Autocorrelation Test Spectral Density Amplitude Estimation Constant Statistics Summary Graphical Comparison
Sensitivity Analysis	temperature adjustment model	Behavior Space Classifier
Parameter Calibration (Pattern based)	temperature adjustment model	Behavior Class Mapper
Policy Design (Pattern based)	temperature adjustment model	Behavior Class Mapper
Policy Analysis (Numerical)	predator-prey model	Model Docking Window Autocorrelation Amplitude Estimation Constant

7. CONCLUSION

System dynamics methodology necessitates formal quantitative analysis methods according to the literature survey on both methodological studies and real life modeling projects. This need applies to all steps in the methodology, including model calibration, model testing, sensitivity analysis, policy analysis and design.

In system dynamics, the purpose of modeling is to understand the causes of real life problems and design policies to obtain improved behaviors. The emphasis is put on pattern reproduction rather than point-prediction. As a result of this characteristic, analysis of outputs of system dynamics models should be done on pattern-oriented basis. Similarity of two dynamic behaviors should be assessed by their pattern related features such as trends, periodicities, fluctuations, amplitudes, etc. This renders traditional statistical curve fitting and forecasting techniques not applicable to behavior analysis.

In the model analysis literature, there exists pattern oriented formal quantitative analysis techniques. However, these tools have not achieved large usage base among the modelers/analysts. Reasons behind this situation are identified as lack of automation of the analysis methods, lack of familiarity with the specialized programming environments, lack of communication with existing model building software and lack of usability. There are example efforts on various levels of automation from proof of concept to packaged software; however they are all affected by the aforementioned problems.

In this thesis available pattern-oriented quantitative behavior analysis methods are reviewed. A group of these methods are selected and a toolkit is formed. The toolkit includes quantitative functions and methods that assess the similarity of behaviors in terms of pattern components and a supervised pattern recognition method that can label a given behavior according to generic dynamic pattern classes. In the second part of the thesis a software is built as a new model behavior analysis environment that

is based on the aforementioned toolkit. The name of the software is BATS (Behavior Analysis and Testing Software). It is written in pure Python language, with extensive utilization of specialized libraries, namely Numpy for the calculations, Matplotlib for plotting and wxPython for graphical user interface. BATS is able to connect with Vensim, a widely used modeling software in the system dynamics field. BATS is able to control a model developed on Vensim by parsing parameters, setting new parameter values, running model and getting results of the simulations. Finally, items for enhancing the user experience that are previously identified are implemented. With the utilization of wxPython library for graphical user interface and Matplotlib for visualization of graphs, it can be stated that the relevant methods and concepts are successfully implemented and a “self-satisfactory” level of user-friendliness is achieved for the software.

The new software is demonstrated in action for various tasks. The step by step instructions provide a guideline for available usage modes of BATS. The demonstration examples include extreme condition testing of the density dependent population growth model, behavior validation of an oscillatory data with trend, behavior sensitivity analysis of the temperature adjustment model, and policy analysis of the predator-prey model.

In summary, final product of this thesis study is a software with graphical user interface. It has data inputting features, data analysis features and model analysis features. Data inputting features include reading from external files, drawing a new data, splitting existing data, extracting selections of data, and two filtering methods, namely exponential smoothing and moving average. Data analysis features include trend regression, autocorrelation, spectral densities, amplitude estimation, cross-correlation, graphical comparison, behavior classification, and summary statistics. Model analysis features include a general purpose model docking window, hypothesis tester, behavior space classifier, and behavior class mapper.

In this thesis there are four main research objectives. First one is to elaborate a pattern-oriented formal analysis methods toolkit. This objective is addressed by

integrating a behavior classification algorithm, namely ISTS algorithm, which is a pattern-based classification method. In addition, a procedural approach called Multi-step Procedure is integration. This set of functions enables extracting more detailed numerical pattern information such as mean, variance, trend, slope, period, amplitude. Thirdly, there is an implementation based graphical inspection procedure, which is flexible and manual.

The second objective is to enable automated analysis at least at a function-level. This is achieved at the function-level in Data Analysis features such as Trend Regression, Autocorrelation, Amplitude Estimation. The more desired automation which is defined as task-level automation is achieved in two model analysis features called Behavior Space Classifier and Behavior Class Mapper. This method is capable of performing task-level automated analysis, namely the task of behavior pattern sensitivity with respect to changes in parameters.

The third objective is to communicate with existing modeling software. In this project currently it is possible to set parameters, send commands and get results from a model developed on Vensim. In future work, it is planned and also possible to implement communication methods for other existing popular modeling software.

The fourth objective is to maintain a “satisfactory” level of usability, which is addressed by implementing state of the art user interface design principles and TDI design, and Python libraries like Matplotlib and wxPython. Future work of this study is to assess the usefulness of the developed software and search for improvements and promotion. Currently usability criteria are self-suggested and self-satisfied. An independent and quantitative inspection of the user-friendliness of BATS would be more accurate and more usability improvements can be made. Nevertheless the program is user-friendly with respect to model formulation and model analysis experience, expertise and expectations of the author and the project stakeholders.

Extensibility of BATS is also partially considered in this thesis. On the developer side there is still a need for familiarity with the programming language and environ-

ment. However, the selected programming language Python, as discussed earlier, offers ease of learning.

It is appropriate to note that BATS is an enabler for user-friendly delivery of future dynamic behavior analysis and model analysis methods. Using its parameter search engine it is possible to develop new specialized model analysis methods. This is applicable to both numerical sensitivity, behavior sensitivity, as well as numeric search and pattern-based search.

Future work related to this study can be grouped into two. First group is related to the user experience. This would be maintained in time as the software is being used and tested. Second group is related to improvements in the pattern oriented analysis toolkit. It is desired to include new analysis functions and methods, both that works at the data level (e.g. autocorrelation, spectral density) and at the model level (e.g. behavior space classifier, behavior space mapper). Among the second group of future works, the immediate beneficial contribution would be enhancement of existing time-series classification algorithm to handle oscillatory patterns in a more efficient way and development of new time-series classification and/or clustering algorithms.

APPENDIX A: ESSENTIAL CODE PARTS

A.1. ISTS Algorithm

```

import numpy as np
from scipy import interpolate

IDS = ["zero0", "const", "plnr", "nlnr", "nexgr",
       "sshgr", "pexgr", "gr1da", "gr1db", "gr2da",
       "gr2db", "d1peg", "d2peg", "nexdc", "sshdc",
       "pexdc", "d1gra", "d1grb", "d2gra", "d2grb",
       "g1ped", "g2ped", "oscct", "oscgr", "oscdc"]

def testhy(y):
    """
    Main function of the module. Returns likelihoods so that
    the table can be used further in shape decision analysis
    and hypothesis testing

    Parameters
    -----
    y : input data

    Returns
    -----
    table: state-optimized likelihood values for each class
    """
    y=changesize(y)
    #Initialize table
    table = -10*np.ones(len(IDS))

    #Test for zero, constant or else
    c = const(y)

    #Zero: Fill zero0 as 10
    if c==1:
        table[0] = 10

    #Constant: Fill constant as 10
    if c==2 or c==3:
        table[1] = 10

    #Non-constant: Calculate table using HMM, classt function
    if c!=1 and c!=2:
        table = classt(y, table)

    return table

def autocorrelation(y):
    lag = int(len(y)/2)
    r_k = []
    for i in range(1, lag+1):
        temp = covariance(y, i)

```

```

        r_k = np.append(r_k, temp)
    return r_k / covariance(y, 0)

def covariance(y, lag):
    temp = 0
    for i in range(len(y)-lag):
        temp += (y[i]-np.mean(y)) * (y[i+lag]-np.mean(y))
    cov = temp / (len(y)-lag)
    return cov

def changesize(y, target=120):
    x = np.arange(len(y))
    f = interpolate.interp1d(x,y)
    xnew = np.linspace(0, len(y)-1, target)
    ynew = f(xnew)
    return ynew

def classt(y, table):
    """
    Extracts training data from the specified txt file ,
    calculates state-optimized likelihoods utilizing
    Viterbi Algorithm according to Hidden Markov Model definitions.
    This is the body of this module where most calculations occur.
    Calls: normalize, feat, viterbi, autocorrelation

    Parameters
    _____
    y : input data
    table : initial table

    Returns
    _____
    table : final table after calculations
    """
    previous_directory = os.getcwd()
    input_file_directory=os.path.dirname(os.path.realpath(sys.argv[0]))
    input_file_directory += "\\data"
    os.chdir(input_file_directory)
    input_file = "ISTS_Training_Data_07_2013.txt"
    f = open(input_file)

    #=====
    # Read Training Parameters
    #=====
    _type = np.array(f.readline(), np.int32)    #1
    _key = np.array(f.readline(), np.int32)    #1
    T = np.array(f.readline(), np.int32)       #num of segments = 12
    N = np.array(f.readline(), np.int32)       #num of states = 7
    nclass = np.array(f.readline(), np.int32)  #num of classes = 25
    source = []
    for _i in range(nclass):
        source = np.append(source, np.array(f.readline(), np.int32))

    #=====
    # Normalization
    # YN1 is for main i's; YN2 is for mirrors
    # YN3 is for plain and YN4 are slope for de-trended oscillations

```

```

#=====
YN1 = normalize(y)
YN2 = 1.0 - normalize(y)

# List of Mirrors
# id: 01 02 03 04 05 06 07 08 09 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25
mr = [ 0, 0, 4, 0, 14, 15, 16, 17, 18, 19, 20, 21, 22, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

#=====
# Main loop
# Iterated i's are [3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 23]
#=====
loop = True
while loop:

    i = np.array(f.readline(), np.int32)

    p = np.matrix(np.empty((N,1)))
    for j in range(N):
        for k in range(1):
            p[j,k] = np.array(f.readline(), np.float64)

    A = np.matrix(np.empty((N,N*(T-1))))
    for j in range(N):
        for k in range(N*(T-1)):
            A[j,k] = np.array(f.readline(), np.float64)

    m = np.matrix(np.empty((3,N)))
    for j in range(3):
        for k in range(N):
            m[j,k] = np.array(f.readline(), np.float64)

    V = np.matrix(np.empty((3,3*N)))
    for j in range(3):
        for k in range(3*N):
            V[j,k] = np.array(f.readline(), np.float64)

    samsz = np.array(f.readline(), np.int32)

    lkd = np.empty(samsz)
    for j in range(samsz):
        lkd[j] = np.array(f.readline(), np.float64)

    st = np.matrix(np.empty((samsz,T)))
    for j in range(samsz):
        for k in range(T):
            st[j,k] = np.array(f.readline(), np.float32)

#=====
# Evaluate Likelihoods
# Calculated i's are [3, 5, 6, 7, 8, 9, 10, 11, 12, 13]
#=====
lkdin, optseq = viterbi(p, A, m, V, feat(YN1, T))
table[i-1] = (lkdin-np.mean(lkd)) / np.std(lkd)

```

```

#=====
# Mirrors
# For calculating patterns which are x-axis symmetric to
# current pattern. Mirror of plnr(3) is nlnr(4), mirror
# of nexgr(5) is nexdc(14), etc.
# See list of mirrors defined above for all mirrors
# Calculated i's are [4, 14, 15, 16, 17, 18, 19, 20, 21, 22]
#=====
mir = mr[i-1]
if mir != 0:
    lkdin, optseq = viterbi(p, A, m, V, feat(YN2, T))
    table[mir-1] = (lkdin-np.mean(lkd)) / np.std(lkd)

#=====
# Oscillatory patterns
# Calculated i's are [23, 24, 25]
#=====
if i == 23:
    YN3, YN4, slope = oscillation_preparation(y)
    lkdin, optseq = viterbi(p, A, m, V, feat(YN3, T))
    table[23-1] = (lkdin-np.mean(lkd)) / np.std(lkd)

    lkdin, optseq = viterbi(p, A, m, V, feat(YN4, T))
    if slope > 0.2:
        table[24-1] = (lkdin-np.mean(lkd)) / np.std(lkd)
    elif slope < 0.2:
        table[25-1] = (lkdin-np.mean(lkd)) / np.std(lkd)

#=====
# End of the loop for an i
#=====
if i == 23:
    loop = False

f.close()
os.chdir(previous_directory)
return table

def const(y):
    band = 0.05
    mean = np.mean(y)
    maxgap = np.max(y) - np.min(y)
    if mean==0 and maxgap==0:
        return 1 #constant with zero mean
    if mean!=0 and maxgap==0:
        return 2 #constant with non-zero mean
    q1 = y <= mean*(1+band)
    q2 = y >= mean*(1-band)
    inband = np.all(q1*q2)
    if maxgap!=0 and inband:
        return 3 #constant within band
    else:
        return 0 #not constant

def feat(y, T):
    """
    Constructs feature vector from input data y for t segments.

```

First segments y into T segments, then calculates the features.

Parameters

y : input row vector of size K
T : number of time segments (currently t=12)

Returns

S : feature matrix of shape (3, t)
- 1st feature: slope of linear fit on the segment
- 2nd feature: curvature at the midpoint of the segment
- 3th feature: segment mean
"""

`K = len(y)        #K=120 T=12`

`segment_length = int(round(K / T))    #segment_length=10`
`segments = y.reshape((T,-1))`

`S = []`

`for t in range(T):`

`#feature 1: slope of linear fit`

`_fit1, coefs = polyfit(segments[t], 1, K)`

`_intercept, slope = coefs`

`slope = np.float64(slope)`

`S = np.append(S, slope)`

`#feature 2: curvature at the midpoint of the segment`

`_fit2, coefs = polyfit(segments[t], 2, K)`

`_c, b, a = coefs`

`b = np.float64(b)`

`a = np.float64(a)`

`midpoint = 0.5 * (segment_length-1) / (K-1)`

`par = b**2 + 4*a*b*midpoint + 4*(a**2)*(midpoint**2)`

`curvature = (2*a) / ((1+par)**(1.5))`

`S = np.append(S, curvature)`

`#feature 3: segment mean`

`segment_mean = segments[t].mean()`

`S = np.append(S, segment_mean)`

`S = S.reshape((-1,3)).T`

`S = np.matrix(S)`

`return S`

`def fpeak(x):`

`"""fpeak(y) finds the first peak in the signal y"""`

`#Get difference between 2 succeeding elements of an array`

`diff = x[1:] - x[:-1]`

`#Get Signum function values -1, 0, 1`

`d = np.sign(diff)`

`pos = 0`

`p = 0`

`i = 1`

```

peak_found = 0

while peak_found == 0 and i<=len(d):
    if pos == 1 and d[i-1] <= 0:
        p = i
        peak_found = 1
    if d[i-1] > 0:
        pos = 1
    i = i + 1
return p

def logl(x):
    rX, cX = x.shape
    for i in range(rX):
        for j in range(cX):
            x[i,j] = np.log(max(1.0e-10, x[i,j]))
    return x

def normalize(y):
    y = (y - np.min(y)) / (np.max(y) - np.min(y))
    return y

def oscillation_preparation(y):
    MIN_SLOPE = 0.2
    MIN_PERIOD = len(y)/1.5
    acf = autocorrelation(normalize(y))
    acfpeak = fpeak(acf)
    if acfpeak < MIN_PERIOD and acfpeak!= 0:
        YN3 = changesize(acf[:acfpeak], len(y))
        YN3 = normalize(YN3)
    else:
        YN3 = np.zeros(len(y))

    fit, coefs = polyfit(normalize(y), 1, len(y))
    fit = np.array(fit)[0]
    _intercept, slope = coefs
    if abs(slope) > MIN_SLOPE:
        resid = normalize(y)-fit
        acf = autocorrelation(normalize(resid))
        if acfpeak < MIN_PERIOD and acfpeak!= 0:
            YN4 = changesize(acf[:acfpeak], len(y))
            YN4 = normalize(YN3)
        else:
            YN4 = np.zeros(len(y))
    else:
        YN4 = np.zeros(len(y))
    return YN3, YN4, slope

def polyfit(x, deg, N):
    """
    Fits a polynomial of order deg to the input vector.
    """
    x = np.matrix(x)
    _cx, rx = x.shape
    t=np.linspace(0.0, (rx-1)/(N-1), rx)
    B = np.matrix([[1.0]*rx]*(deg+1))
    for i in range(1,deg+1):

```

```

        B[i,:] = np.power(t,i)
    coef = (B*B.T).I * B * x.T
    fit = coef.T * B
    return fit , coef

```

```

def viterbi(p, A, m, V, S):

```

```

    """

```

```

    Implementation of Viterbi Algorithm for dynamic
    pattern classification. Calculates the state-optimized
    likelihood function and the optimal sequence for a
    non-stationary HMM model.

```

```

    Input Parameters

```

```

    p : initial probability column vector of shape (1, N)
    A : state transition matrix from segment t to t+1 of
        shape (N, N*(T-1))
    m : mean vector of states of shape (3, N)
    V : covariance matrix of states of shape (3, N*3)
    S : observation vector
        In ISTS, S is the feature vector of shape (3, T)

```

```

    Function Parameters

```

```

    N : number of states (in current version N=7)
    T : number of time segments (in current version T=12)
    B : Set of posteriori density of observation vectors in states
        of shape (N, T)
        b_j(o_t) : observation vector o_t in state j.

```

```

    Returns

```

```

    lkd : state-optimized likelihood function
    optseq : optimum state sequence
    """

```

```

    N, _nt1 = A.shape

```

```

    rS, T = S.shape

```

```

    #=====
    # Density Function Values
    # Calculate B matrix
    #=====

```

```

    B = []

```

```

    for i in range(N):

```

```

        C = V[:, i*rS:(i+1)*rS]

```

```

        constant = 1 / ( ((2*np.pi)**rS) * np.sqrt(np.linalg.det(C)) )

```

```

        for t in range(T):

```

```

            omm = S[:, t] - m[:, i]

```

```

            b_it = constant * np.exp(-0.5 * omm.T * np.linalg.inv(C) * omm)

```

```

            B = np.append(B, b_it)

```

```

    B = np.reshape(B, (N, T))

```

```

    B = np.matrix(B)

```

```

    #=====

```

```

# Logarithmic Conversions
#=====
logp = logl(np.copy(p))
logA = logl(np.copy(A))
logB = logl(np.copy(B))

#=====
# Viterbi Algorithm Step 1: Initialization
#=====
#This is the first time we use this initialization method
d = np.matrix([[0.0]*N]*T)
psi = np.matrix([[0.0]*N]*T)
# i.e. t=0
d[0] = logp.T + logB[:,0].T

#=====
# Viterbi Algorithm Step 2: Recursive Computation
#=====
for t in range(1,T):
    for j in range(0,N):
        dd = d[t-1] + (logA[:,j+(t-1)*N]).T + logB[j, t]
        d[t, j] = np.max(dd)
        psi[t, j] = np.argmax(dd)

#=====
# Viterbi Algorithm Step 3: Termination
#=====
lkd = np.max(d[T-1])
optseq = np.array([0]*T)
optseq[T-1] = np.argmax(d[T-1])
for t in range(T-2, -1, -1):
    optseq[t] = psi[t+1, optseq[t+1]]

return lkd, optseq

```

APPENDIX B: EQUATIONS OF TEST MODELS

B.1. Density Dependent Growth Model

```

bf = normal bf*effect of crowding on bf
births = Population*bf
Capacity = 200
Crowding = Population/Capacity
deaths = Population*df
df = 0.06
effect of crowding on bf= WITH LOOKUP (Crowding,
    [(0,0)-(2,1.33)],(0,1.33333),(0.166667,1.31667),
    (0.333333,1.3),(0.5,1.25),(0.666667,1.18333),
    (0.833333,1.1),(1,1),(1.16667,0.883333),(1.33333,
    0.745),(1.5,0.583333),(1.66667,0.4),(1.83333,
    0.211667),(2,0) ))
FINAL TIME = 300
    The final time for the simulation.
initial population = 20
INITIAL TIME = 1
    The initial time for the simulation.
normal bf = 0.06
Population= INTEG (births-deaths, initial population)
SAVEPER = 1
    The frequency with which output is stored.
TIME STEP = 0.125
    The time step for the simulation.

```

B.2. Temperature Adjustment Model

```

Actual Room Temperature = INTEG (
    Room Temperature Change, -10)
Desired Room Temperature = 0
FINAL TIME = 100
    The final time for the simulation.
INITIAL TIME = 0
    The initial time for the simulation.
Perceived Room Temperature= INTEG (
    Perception Change, 5)
Perception Change =
    (Actual Room Temperature - Perceived Room Temperature) /
    Perception Delay
Perception Delay = 10

```

```

Room Temperature Change=
    (Desired Room Temperature - Perceived Room Temperature) /
    Temperature Adjustment Delay
SAVEPER = 1
    The frequency with which output is stored.
Temperature Adjustment Delay = 10
TIME STEP = 0.125
    The time step for the simulation.

```

B.3. Predator-Prey Model

```

alpha = 0.15
beta = 0.003
birthsX = X*alpha
birthsY = contacts*delta
contacts = X*Y
cX = 1.25 [-1,2,0.001]
cY = 1.25 [0,2,0.01]
deathsX = contacts*beta
deathsY = Y*gamma
delta = 0.001
FINAL TIME = 200
    The final time for the simulation.
gamma = 0.1
harvestsX = X*hx
harvestsY = Y*hy
hx = 0.05
hy = 0.1
init X = cX*gamma/delta
init Y = cY*alpha/beta
INITIAL TIME = 0
    The initial time for the simulation.
SAVEPER = 1
    The frequency with which output is stored.
TIME STEP = 0.0625
    The time step for the simulation.
X = INTEG (birthsX-deathsX-harvestsX, 250)
Y = INTEG (birthsY-deathsY-harvestsY, 40)

```

REFERENCES

- Anderson, O. D., 1982, "Sample Serial Correlations From ARIMA Processes," in D.Anderson, O. and M. R.Perryman. (eds.), "In Applied Time Series Analysis," pp. 1131 – 1175, Netherlands, North-Holland.
- Arenas, F., 2005, "An Adaptive Expectations Approach to the Mechanisms of Transmission Model of the Central Bank of Colombia," *Proceedings of the 23rd International Conference of the System Dynamics Society*, The System Dynamics Society, Boston.
- Bagnall, A., C. Ratanamahatana, E. Keogh, S. Lonardi and G. Janacek, 2006, "A Bit Level Representation for Time Series Data Mining with Shape Based Similarity," *Data Mining and Knowledge Discovery*, Vol. 13, No. 1, pp. 11–40.
- Barlas, Y., 1989, "Multiple Tests for Validation of System Dynamics Type of Simulation Models." *European Journal of Operational Research*, Vol. 42, No. 1, pp. 59–87.
- Barlas, Y., 1990, "An Autocorrelation Function Test For Output Validation." *Simulation*, Vol. 55, No. 1, pp. 7–16.
- Barlas, Y., 1996, "Formal Aspects of Model Validity and Validation in System Dynamics." *System Dynamics Review*, Vol. 12, No. 3, pp. 183–210.
- Barlas, Y., 2002, "System Dynamics: Systemic Feedback Modeling for Policy Analysis," in Barlas, Y. (ed.), "Knowledge for Sustainable Development - An Insight into the Encyclopedia of Life Support Systems," pp. 1131 – 1175, UNESCO-EOLSS Publishers, Paris, France; Oxford, UK.
- Barlas, Y., 2007, "Leverage points to march "upward from the aimless plateau"," *System Dynamics Review*, Vol. 23, No. 4, pp. 469–473.
- Barlas, Y. and A. Erdem, 1994, "Output Behavior Validation In System Dynamics

- Simulation,” *Proceedings Of The European Simulation Symposium*, İstanbul, Turkey, pp. 81–84.
- Barlas, Y. and K. Kanar, 1999, “A Dynamic Pattern-oriented Tests for Model Validation,” *Proceedings of 4th Systems Science European Congress*, İstanbul, Turkey, The System Dynamics Society.
- Barlas, Y., H. Topaloğlu and S. Yılkaya, 1997, “A Behavior Validity Testing Software (BTS),” *Proceedings of 15th International Conference of the System Dynamics Society*, İstanbul, Turkey, The System Dynamics Society.
- Bass, F. M., 1969, “A New Product Growth for Model Consumer Durables,” *Management Science*, Vol. 15, No. 5, pp. 215–227.
- Bloomfield, P., 1976, *Fourier Analysis of Time Series an Introduction*, Wiley, New York.
- Boğ, S. and Y. Barlas, 2005, “Automated Dynamic Pattern Testing, Parameter Calibration and Policy Improvement,” *Proceedings of the 23rd International Conference of the System Dynamics Society*, boston, MA, USA., The System Dynamics Society.
- Bowerman, B. L., R. T. O’Connell and A. B. Koehler, 2005, *Forecasting, Time Series, and Regression: An Applied Approach*, Thomson Brooks/Cole, Belmont, CA.
- Box, G. and G. Jenkins, 1970, *Time Series Analysis, Forecasting and Control*, Holden-Day Inc., San Francisco, CA.
- Bozyayla, E., 2001, *Evaluation of Different Model Behavior Validation Tests*, M.S. Thesis, Boğaziçi University.
- Drechsler, M., 1998, “Sensitivity Analysis of Complex Models,” *Biological Conservation*, Vol. 86, No. 3, pp. 401 – 412.
- Ekşin, C., 2009, *Application of Genetic Algorithms to Analysis and Policy Design in System Dynamics*, M.S. Thesis, Boğaziçi University.

- EMA Team, 2011, *Exploratory Modelling and Analysis (EMA) Workbench*, URL "<http://simulation.tbm.tudelft.nl/ema-workbench/contents.html>", accessed at December 2013.
- Eskinasi, M., E. Rouwette and J. Vennix, 2009, "Simulating Urban Transformation in Haaglanden, the Netherlands," *System Dynamics Review*, Vol. 25, No. 3, pp. 182–206.
- Forrester, J. and P. Senge, 1980, "Test for Building Confidence in System Dynamics Models," *TIMS Studies in the Management Sciences*, Vol. 14, pp. 209–228.
- Fu, T., 2011, "A Review on Time Series Data Mining," *Engineering Applications of Artificial Intelligence*, Vol. 24, No. 1, pp. 164 – 181.
- Ghaffarzadegan, N. and A. T. Tajrishi, 2010, "Economic Transition Management in a Commodity Market: The Case of the Iranian Cement Industry," *System Dynamics Review*, Vol. 26, No. 2, pp. 139–161.
- Granger, C. W. I. and M. Hatanaka, 1964, *Spectral Analysis of Economic Time Series*, Princeton University Press, Princeton, NJ.
- Güneralp, B., 2006, "Towards Coherent Loop Dominance Analysis: Progress in Eigenvalue Elasticity Analysis." *System Dynamics Review*, Vol. 22, No. 3, p. 263–289.
- Hearne, J., 1985, "Sensitivity Analysis of Parameter Combinations," *Applied Mathematical Modelling*, Vol. 9, No. 2, pp. 106 – 108.
- Hoover, S. and R. Perry, 1989, *Simulation: A Problem-Solving Approach*, Addison-Wesley, Boston.
- Huang, N. and N. Attoh-Okine, 2005, *The Hilbert-Huang Transform in Engineering*, Taylor & Francis, Florida.
- Huang, N. E., Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N.-C. Yen, C. C. Tung and H. H. Liu, 1998, "The Empirical Mode Decomposition and the Hilbert

- Spectrum for Nonlinear and Non-stationary Time Series Analysis,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, Vol. 454, No. 1971, pp. 903–995.
- Huber, P. J., 2011, *Data Analysis What Can Be Learned From the Past 50 Years*, John Wiley & Sons Inc, Hoboken, NJ.
- Hunter, J. D., 2007, “Matplotlib: A 2D Graphics Environment,” *Computing In Science & Engineering*, Vol. 9, No. 3, pp. 90–95.
- Johnson, J., 2008, *GUI bloopers 2.0: Common User Interface Design Don'ts and Dos*, Elsevier/Morgan Kaufmann, Amsterdam/Boston.
- Kampmann, C. E. and R. Oliva, 2006, “Loop Eigenvalue Elasticity Analysis: Three Case Studies,” *System Dynamics Review*, Vol. 22, No. 2, pp. 141–162.
- Kanar, K., 1999, *Structure Oriented Behaviour Tests in Model Validation*, M.S. Thesis, Boğaziçi University.
- Keogh, E. and S. Kasetty, 2002, “On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration,” *SIGKDD'02*, pp. 102–111.
- Koopmans, L. H., 1974, *The Spectral Analysis of Time Series*, Academic Press, New York.
- Krakenes, A., 2004, “SOPS—a Tool to Find Optimal Policies in Stochastic Dynamic Systems,” Tech. rep., WPSD 2/05, System Dynamics Group, University of Bergen, <http://www.ifi.uib.no/sd/wp.html>, accessed at December 2013.
- Langtangen, H. P., 2009, *Python Scripting for Computational Science*, Springer, Berlin, 3rd edn.
- Liao, T. W., 2005, “Clustering of Time Series Data — A Survey,” *Pattern Recognition*, Vol. 38, pp. 1857–1874.

- Mahajan, V., E. Muller and F. M. Bass, 1990, “New Product Diffusion Models in Marketing: A Review and Directions for Research,” *Journal of Marketing*, Vol. 54, No. 1, pp. 1–26.
- Martinez-Moyano, I. J. and G. P. Richardson, 2013, “Best Practices in System Dynamics Modeling,” *System Dynamics Review*, Vol. 29, No. 2, p. 102–123.
- Miller, J. H., 1998, “Active Nonlinear Tests (ANTs) of Complex Simulation Models,” *Management Science*, Vol. 44, No. 6, pp. 820 – 830.
- Mills, T. C., 2011, *The Foundations of Modern Time Series Analysis*, Palgrave Advanced Texts in Econometrics, Palgrave Macmillan, New York.
- Mitsa, T., 2010, *Temporal Data Mining*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Taylor & Francis, Boca Raton, FL.
- Moxnes, E., 2005, “Policy Sensitivity Analysis: Simple Versus Complex Fishery Models,” *System Dynamics Review*, Vol. 21, No. 2, pp. 123—145.
- Nielsen, J., 1993, *Usability Engineering*, Morgan Kaufmann, Boston.
- Oliphant, T. E., 2007, “Python for Scientific Computing,” *Computing in Science Engineering*, Vol. 9, No. 3, pp. 10–20.
- Otto, P. and M. Simon, 2008, “Dynamic Perspectives on Social Characteristics and Sustainability in Online Community Networks,” *System Dynamics Review*, Vol. 24, No. 3, pp. 321–347.
- Percival, D. and A. Walden, 2000, *Wavelet Methods for Time Series Analysis*, Cambridge series on statistical and probabilistic mathematics, Cambridge University Press, Cambridge, New York.
- Pérez, F., B. Granger and J. Hunter, 2007, “Python: An Ecosystem for Scientific Computing,” *Computing in Science & Engineering*, Vol. 13, No. 2, pp. 13–21.

- Peter J. Brockwell, R. A. D., 2006, *Time Series: Theory and Methods*, Springer, New York.
- Peterson, D. W. and R. L. Eberlein, 1994, "Reality Check: A Bridge Between Systems Thinking and System Dynamics." *System Dynamics Review*, Vol. 10, No. 2-3, pp. 159–174.
- Phaff, W. G., 2006, *Investigating Model Analysis: A Critical Examination of Current Methods for Model Behavioural Analysis in System Dynamics*, M.S. Thesis, Delft University of Technology.
- Pollock, D. S. G., 1987, "The Methods of Time-Series Analysis," *Interdisciplinary Science Reviews*, Vol. 12, No. 2, pp. 128–135.
- Prechelt, L., 2003, "Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java," Vol. 57 of *Advances in Computers*, pp. 205 – 270, Elsevier.
- Richardson, G. P., 1996, "Problems for the Future of System Dynamics," *System Dynamics Review*, Vol. 12, No. 2, pp. 141–157.
- Richardson, G. P., 1999, "Reflections for the Future of System Dynamics," *The Journal of the Operational Research Society*, Vol. 50, No. 4, pp. 440–449.
- Scharf, L. and C. Demeure, 1991, *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*, Addison-Wesley Series in Electrical and Computer Engineering, Addison-Wesley Pub. Co., Reading, MA.
- Shannon, R. E., 1975, *Systems Simulation: The Art and Science*, Prentice Hall, Englewood Cliffs, NJ.
- Shasha, D. and Y. Zhu, 2004, *High Performance Discovery in Time Series: Techniques and Case Studies*, Monographs in Computer Science, Springer, New York.

- Soylu, S., 2006, *Generic Dynamic Patterns: Testing by Empirical Evidence*, M.S. Thesis, Boğaziçi University.
- Sterman, J., 1984, "Appropriate Summary Statistics for Evaluating the Historic Fit of System Dynamics Models," *Dynamica*, Vol. 10, pp. 51–66.
- Sterman, J. D., 2000, *Business Dynamics*, McGraw-Hill, New York.
- Theil, H., 1966, *Applied Economic Forecasting*, Studies in Mathematical and Managerial Economics, North-Holland Pub. Co., Amsterdam.
- Tsay, R. S., 2000, "Time Series and Forecasting: Brief History and Future Research," *Journal of the American Statistical Association*, Vol. 95, No. 450, pp. 638–643.
- Tsay, R. S., 2002, *Analysis of Financial Time Series*, Wiley Series in Probability and Statistics, Wiley, New York.
- Yücel, G. and Y. Barlas, 2011, "Automated Parameter Specification in Dynamic Feedback Models Based on Behavior Pattern Features." *System Dynamics Review*, Vol. 27, No. 2, pp. 195–215.
- Yule, G. U., 1927, "On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, Vol. 226, pp. 267–298.