

**SYMBOLIC MATLAB PACKAGE FOR NONLINEAR CONTROL
DESIGN**

BY
Khalid E. Al-Khater

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

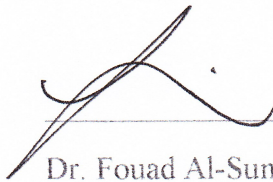
December 2014

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS


DHAHRAN- 31261, SAUDI ARABIA

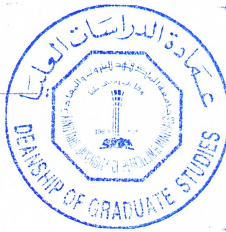
DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Khalid E. Al-Khater** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SYSTEMS ENGINEERING**.

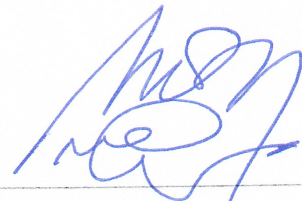

Dr. Fouad Al-Sunni


Department Chairman



Dr. Salam A. Zummo
Dean of Graduate Studies



24/2/15
Date


Dr. Magdi Mahmoud
(Advisor)


Dr. Fouad Al-Sunni
(Member)


Dr. Abdul Wahid Al-Saif
(Member)

© Khalid E. Al-Khater

2014

I lovingly dedicate this thesis to my parents, my wife, and my friends who supported me
each step of the way.

ACKNOWLEDGMENTS

Thanks are due to Almighty Allah for the knowledge and diligence that he has granted upon me during this research project, and indeed, throughout my life. Also, I would like to thank my parents and my wife for their unconditional support throughout my degree. In particular, the patience and understanding shown by my parents and my wife during the graduate study is greatly appreciated. My fellow brothers and friends are also appreciated for their continual support and encouragement throughout this work specially Mohammed Yaaqoub, Majdi Al-Basarah, Ali Al-Malak, Ahmed Al-Shaer and Ahmed Saleem.

Special thanks to Dr. Magdi Mahmoud, for making this research possible. His support, guidance, advice throughout the research project, as well as his pain-staking effort in reading the drafts, are greatly appreciated. Indeed, without his guidance, I would not be able to put the topic together. Thanks Dr. Magdi and committee members Dr. Fouad Al-Sunni and Dr. Abdul-Wahid Al-Saif.

Finally, I would also like to thank the Deanship of scientific Research (DSR) at KFUPM for financial support through the research group project RG-1316. I would also like to appreciate the support, facilities, and people at KFUPM that made this work possible.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	V
TABLE OF CONTENTS	VI
LIST OF TABLES.....	X
LIST OF FIGURES.....	XI
LIST OF ABBREVIATIONS AND NOMENCLATURE.....	XII
ABSTRACT.....	XIII
ملخص الرسالة.....	XIV
CHAPTER 1: INTRODUCTION	1
1.1 Introduction.....	1
1.2 Why Nonlinear Control?.....	1
1.3 Symbolic Computation in Nonlinear Control	3
1.4 Thesis Objectives	4
1.5 Thesis Organization.....	4
CHAPTER 2: LITERATURE REVIEW	5
2.1 Introduction.....	5
2.2 Nonlinear Control	5
2.3 Symbolic Computation in Control Systems.....	6
CHAPTER 3: MATLAB.....	9
3.1 Introduction.....	9
3.2 MATLAB Overview	9

3.3	Symbolic Toolbox	11
3.3.1	What MATLAB Symbolic Toolbox can do?	11
CHAPTER 4: INPUT-OUTPUT EXACT LINEARIZATION		15
4.1	Introduction	15
4.2	Affine and Non-affine Nonlinear Systems	15
4.3	The Relative Degree	17
4.4	Dynamic Extension	20
4.5	State Transformation	21
4.6	Feedback Linearizing Control Law	22
4.7	Exact linearization for non-square systems	24
CHAPTER 5: THE NLC PACKAGE		25
5.1	Introduction	25
5.2	Package Overview and Features	25
5.3	Converting to affine, "nlc_affine" subroutine	27
5.4	Relative Degree Calculation, "nlc_RelDeg" subroutine	28
5.5	Decoupling and output derivatives matrices, "nlc_decoupling" subroutine	29
5.6	Dynamic Extension, "nlc_dynamicExt" subroutine	30
5.7	State Transformation, "nlc_StatTrans" subroutine	32
5.8	Feedback linearizing controller, "nlc_FBL" subroutine	32
5.9	Simulation code generation, "nlc_WriteToFile" subroutine	33
5.10	Examples	35
5.10.1	Contrived Model.....	35
5.10.2	Dynamic Extension Example.....	43
5.10.3	Robot with Flexible Joint	50
5.10.4	Continues Stirred Reactor Model (non-square)	59
5.10.5	Polymerization Fluidized Bed Reactor Model.....	67
CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS		88

6.1	Conclusions	88
6.2	Recommendations and Future Work.....	88
	REFERENCES.....	90
	APPENDIX A: NLC PACKAGE FLOW CHART.....	94
	APPENDIX B: PROGRAM CODES OF NLC SUBROUTINES.....	96
B.1	MAIN PROGRAM "NLC_MAIN.M"	96
B.2	FBR MODEL WITH FIRST ORDER HEAT EXCHANGER.....	100
B.3	FBR MODEL WITH STAGED HEAT EXCHANGER	101
B.4	CSTR MODEL	102
B.5	INDUCTION MOTOR MODEL.....	103
B.6	USER DEFINED MODEL.....	104
B.7	GENERAL NONLINEAR MODEL EXAMPLES	107
B.8	AFFINE CHECK AND AFFINE CONVERSION "NLC_AFFINE.M" & "NLC_MAKEAFFINE.M"	109
B.9	RELATIVE DEGREE COMPUTATION PROGRAM "NLC_RELDEG" AND LIE DERIVATIVE "NLC_LIEDERIVATIVE"	110
B.10	DECOUPLING AND OUTPUT DERIVATIVES MATRICES "NLC_DECOUPLING"	111
B.11	STATE TRANSFORMATION PROGRAM "NLC_STATTRANS.M"	112
B.12	FEEDBACK LINEARIZING CONROL LAW COMPUTATION "NLC_FBL.M" ..	113
B.13	LINEARIZED STATE SPACE SYSEM " NLC_EQULINSYS.M"	114

B.14 SIMULATION CODE GENERATOR "NLC_WRITETOFILE.M"	115
B.15 DYNAMIC EXTENSION PROGRAM "NLC_DYNAMICEXT.M"	119
B.16 PLOT SIMULATION CODE "NLC_PLOTSIMULATION.M"	121
VITAE	122

LIST OF TABLES

Table 1 List of all symbols and algebraic equations for FBR model.....	69
--	----

LIST OF FIGURES

Figure 1 Demonstration of Dynamic Extension.....	21
Figure 2 Feedback linearization interpretation	23
Figure 3 LQI interpretation.....	34
Figure 4 Simulation of Exampale 5.10.1	42
Figure 5 Simulation of the ouput for the system in Example 5.10.1	42
Figure 6 Simulation results for the system of example 5.10.2.....	50
Figure 7 Simulation Results of Example 5.10.3	59
Figure 8 Simulation Results of Example 5.10.4 (Output 1)	66
Figure 9 Simulation Results of Example 5.10.4 (Output 2)	67

LIST OF ABBREVIATIONS AND NOMENCLATURE

NLC	Non-Linear Control (Software package)
LQR	Linear Quadratic Regulator
LQI	Linear Quadratic Integral Control
FBR	Fluidized Bed Reactor
CSTR	Continuous Stirred Reactor
$L_f h(x)$	lie derivative along the vector field f
MIMO	Multiple Input Multiple Output
SISO	Single Input Single Output
FBL	Feed-Back Linearization
u	Vector containing system inputs
x	Vector containing the state variables
h	Vector containing system outputs
\dot{x}	Time derivative
$\ \cdot\ $	Norm 2 operator
$\frac{\partial}{\partial x}$	Partial derivative with respect to x

ABSTRACT

Full Name : Khalid .E Al-Khater
Thesis Title : Symbolic MATLAB Package for Nonlinear Control Design
Major Field : Systems Engineering
Date of Degree : Dec, 2014

The area of nonlinear control has received significant consideration during the past decades because of the insight that linear controllers are insufficient even for moderately nonlinear processes and the availability of new powerful tools. In addition, several symbolic software packages have been developed within various mathematical languages to solve various control problems.

However, complex systems like nonlinear chemical processes that are consuming calculations have led to the need of additional aid, and therefore, development of specific user-friendly computer software/package that can considerably streamline the research. In addition, such software helps finding the gaps in the existing theory and/or verifying its correctness. In the literature quite a lot of methods are proposed for the exact/ analytical solutions of analysis and design problems formulated earlier.

In this thesis, A MATLAB package, which contains a set of symbolic procedures, is introduced and further described. The functions/subroutines available in the package are aimed at the analysis and the design of nonlinear control systems for which it is called “NLC”. The package is implemented utilizing the symbolic MATLAB toolbox. The contribution of this research is as follows: it introduces an extension to the previous MATLAB package intended to compute exact linearizing controller for square, affine and well-defined relative degree systems. The extension covers dynamic extension algorithm to achieve full order relative degree and covers non-affine and non-square systems.

ملخص الرسالة

الاسم الكامل: خالد عيسى مهدي الخاطر

عنوان الرسالة: حزمة برمجية لتصميم المتحكمات الغير خطية

التخصص: هندسة النظم

تاريخ الدرجة العلمية: ديسمبر، ٢٠١٤

يلقى مجال التحكم الغير الخطي اهتماماً كبيراً خلال العقود الماضية بسبب أن التحكم الخطي غير كافٍ حتى بالنسبة للأنظمة الغير الخطية بشكل معتدل، وكذلك بسبب توفر أدوات جديدة وقوية. بالإضافة إلى ذلك، تم تطوير عدة حزم برمجية رمزية بمختلف اللغات البرمجية لحل مشاكل التحكم المختلفة.

لقد أدت الأنظمة المعقدة مثل العمليات الكيميائية الغير خطية والتي تستهلك حساباتها إلى الحاجة لمساعدات إضافية، وبالتالي، تطوير برامج مخصصة أو حزمة سهلة الاستخدام التي تؤدي إلى تبسيط كبير في البحث. هذه الحزم البرمجية تساعد في العثور على الثغرات في النظريات القائمة والتحقق من صحتها. هناك أبحاث كثيرة تقترح الكثير من طرق الحلول التحليلية والرمزية لمشاكل تحليل وتصميم أنظمة التحكم وضعت في وقت سابق.

في هذه الرسالة، يتم عرض حزمة ماتلاب (بالإنجليزية: MATLAB, Matrix-Laboratory) (مختبر المصفوفات) وهو برنامج رائد في التطبيقات الهندسية والرياضية. هذه الحزمة تحتوي على مجموعة من العمليات الرمزية، وسوف يتم وصفها بشكل دقيق في البحث. تهدف البرمجيات الفرعية المتاحة في هذه الحزمة إلى تحليل وتصميم أنظمة التحكم الغير الخطي، ولهذا تسمى "NLC" (نسبة إلى العبارة الإنجليزية Non-Linear Control) ويتم تنفيذ هذه الحزمة باستخدام الأدوات الرمزية في الماتلاب.

يساهم هذا البحث في امتداد حزمة الماتلاب السابقة والمستخدمه حالياً لحساب التحكم الغير خطي للأنظمة المربعة والتألفية والأنظمة ذات الدرجة النسبية واضحة المعالم. هذا الامتداد يشمل:

- "خوارزمية التمديد الديناميكي" لتحقيق الدرجة النسبية الكاملة.
- الأنظمة الغير متألفة والغير مربعة.

CHAPTER 1: INTRODUCTION

1.1 Introduction

Symbolic computation is an evolving practice that can be beneficially used in various areas of applied mathematics and engineering. One of those areas is nonlinear control system analysis and design. A MATLAB package, which contains a set of symbolic procedures, is going to be introduced and further described. The functions/ subroutines available in the package are aimed at the analysis and the design of control systems for which it is called “NLC”. The package is implemented utilizing the symbolic MATLAB toolbox. Other approaches and tools to use symbolic computation in this area of engineering are also known. The contribution of this research is as follows: it introduces an extension to the previous MATLAB package intended to compute exact linearizing controller for square, affine and well-defined relative degree systems. The extension covers dynamic extension algorithm to achieve full order relative degree and covers non-affine and non-square systems.

This chapter is organized as follows: Section 1.2 explains the benefits of nonlinear control methods in the field of control systems design. In section 1.3, the importance of symbolic computation methods in Nonlinear Control is addressed and the motivation this research is highlighted. Section 1.4 states the objectives of this thesis. Finally, the thesis organization is presented in section 1.5.

1.2 Why Nonlinear Control?

There have been remarkable developments in linear control theory. Indeed, it might be reasonably claimed that this topic has been basically resolved. Moreover, linear theory has been widely used in applications. It is not uncommon to find practical systems that utilize very sophisticated linear controllers, e.g., based on high order Kalman filters and LQR theory. However, the real world behaves in a nonlinear fashion – at least when considered over wide operating ranges. Though this

observation, linear control design methods have been successful and hence it must be true that many systems can be well approximated by linear models. In spite of the good performance of linear controllers in many applications, over the last few years the more general topic of nonlinear control has attracted substantial research interest from such broad areas as aircraft control, robotics, process control, and biomedical engineering. This interest due to many reasons such as improvement of existing control systems, analysis of hard nonlinearities, dealing with model uncertainties, and design simplicity.

Linear control methods rely on the key assumption of small range operation for the linear model to be valid. When the required operation range is large, a linear controller is likely to perform very poorly or to be unstable, because the nonlinearities in the system cannot be properly compensated for. Nonlinear controllers, on the other hand, may handle the nonlinearities in large range operation directly. Another assumption of linear control is that the system model is indeed linearizable. However, in control systems there are many nonlinearities whose discontinuous nature does not allow linear approximation. These so-called "hard nonlinearities" include Coulomb friction, saturation, dead-zones, backlash, and hysteresis, and are often found in control engineering. Their effects cannot be derived from linear methods, and nonlinear analysis techniques must be developed to predict a system's performance in the presence of these inherent nonlinearities.

In designing linear controllers, it is usually necessary to assume that the parameters of the system model are reasonably well known. However, many control problems involve uncertainties in the model parameters. A linear controller based on inaccurate or obsolete values of the model parameters may exhibit significant performance degradation or even instability. Nonlinearities can be intentionally introduced to the controller part of a control system so that model uncertainties can be tolerated. Good nonlinear control designs may be simpler and more intuitive than their linear counterparts. This result comes from the fact that nonlinear controller designs are often deeply rooted in the physics of the plants. Linear control may require high quality actuators and sensors to produce linear

behavior in the specified operation range, while nonlinear control may permit the use of less expensive components with nonlinear characteristics. Thus, the subject of nonlinear control is an important area of automatic control.

For highly nonlinear systems, controller design approaches based directly on nonlinear system models can be expected to provide significantly improved performance. During the last decade important progress has been made in the development of such systematic design methodologies for nonlinear control systems. Some of the most promising controller techniques are based on exact linearization theory. Unlike Jacobian linearization, which linearizes the system only at the nominal working point, these methods employ nonlinear transformations and feedbacks that provide exact linearization of the model. The nonlinear model can be linearized under some quite strict conditions. Once a linear model is available, linear controller design techniques can be employed in order to satisfy additional control objectives. Exact linearization theory will be explained further in chapter-4.

1.3 Symbolic Computation in Nonlinear Control

Complex systems like nonlinear chemical processes that are consuming calculations have led to the need of additional aid, and therefore, development of specific user-friendly computer software/package that can considerably streamline the research. In addition, such a software help finding the gaps in the existing theory and/or verifying its correctness. In the literature quite a lot of methods are proposed for the exact/ analytical solutions of analysis and design problems formulated earlier. For small scale test problem these approaches require already a fair amount of computation. For industrial scale problems these methods can only be applied when the computations are assisted by Symbolic Computation programs. This is of specific scientific concern because numerical solutions often gives only an approximation of the properties of the system (along a trajectory or in a working point). At the moment, there is little (industrial) experience with these new controller design strategies due to this computational problems and the fact that they were just recently developed.

However, results established so far in mechanical and process control, seems to be promising, though many problems are yet to be solved. More research is required, in which more realistic models should be used and experimental studies should be conducted. It is expected that the introduction of symbolic computation will be of great consequence to this future research.

1.4 Thesis Objectives

The goal of this research is to present and describe a specific MATLAB-based symbolic computational package (NLC). This is a software project addressing Non-Linear Control systems. The focus will be on Input-Output exact linearization for nonlinear SISO and MIMO systems (affine and non-affine, square and non-square). It is also aimed to build up a user-friendly MATLAB package that can be used for nonlinear controller design and to ensure effectiveness of the package via testing of several nonlinear systems. New contributions and list of drawbacks as well as suggested improvements and future works to be addressed.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter-2 will give a literature review while chapter-3 will present an overview of MATLAB and MATLAB symbolic toolbox. In chapter-4, theory of exact linearization will be explained in details. The developed NLC package will be covered in details in chapter-5. Finally, the conclusion and future research areas will be discussed in the sixth chapter.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

Various methods for nonlinear control has been proposed in the literature. Many of those control methods have been applied utilizing symbolic computation software. This chapter is showing an overview of literatures. It is divided into two sections. The first section covers nonlinear control literature review and the second section is about control methods using symbolic computation.

2.2 Nonlinear Control

The area of nonlinear control has received significant consideration during the past decades because of the insight that linear controllers are insufficient even for moderately nonlinear processes and the availability of new powerful tools. There have been several review articles aiming at giving a view on the present status of the area and trace further directions (McLellan et al., 1990; Roger Brockett, 2014). So far, the two major research directions have been the model-predictive approach and the geometric approach. The key advantage of model-predictive control (MPC) has been its attractive in-built explanation, whereas the key advantage of the geometric approach has been its solid theoretical justification. Geometric process control methods have evolved after about a decade of research on the mathematical characteristics of continuous-time nonlinear systems, using techniques from differential geometry. The system theoretic properties of continuous-time nonlinear systems become well-understood (Isidori, 1989; Khalil, 1996), and they provide the theoretical foundations for nonlinear controller design.

In 1980s, the fundamental differential geometry for nonlinear systems was developed fast. The modern nonlinear control theory was formed. The formation of modern nonlinear control theory not only expand the linear control theory, but also boost the theory's application and research in all kinds of the complex nonlinear control objects. Exact linearization is one of the most important outcomes. This kind of feedback linearization, which is based on differential geometry and different from the

traditional Taylor expansion formula in local linearization, is more precise as a whole, because none higher order nonlinear term is ignored in the linear progress. After decades of development, the nonlinear differential geometry control not only has been a whole system in theory, but also has been widely applied to engineering control, such as in the power electronic system, robots, and chemical processes. Kravaris and Soroush (1990) utilized the I/O linearization framework for multi-input multi-output (MIMO) systems in conjunction with an external multivariable controller with integral action to derive a globally linearizing controller (GLC). The proposed control methodology is tested through simulations in a semi-batch copolymerization reactor. Tarn (1984) proposed nonlinear feedback control for implementation of an advanced dynamic control strategy for robot arms. An algorithm is given for the construction of the required nonlinear feedback. To design a dynamic control for robot arms the above result applied to the JPL-Stanford arm and proposed a new control strategy, which also contains an optimal error-correcting feedback. Simulation results show great promise for the obtained dynamic control strategy. Guay and McLellan (1994) developed algorithm to treat input output linearization for general nonlinear systems. SCHWARTZ (1995) provided a parameter-independent method of achieving full vector relative degree for nonlinear multivariable systems which do not have it. Karimi et al. (2006) proposed a robust feedback linearization scheme based on Lyapunov function in order to cope the model uncertainties of a non linearizable MIMO nonlinear System. The suggested technique was applied to a Twin Rotor system. Bennoune et al. (2007) applied feedback linearization techniques to the problem of maximizing the efficiency of the induction motor by minimizing the energy losses. Next section will give a survey of the symbolic computations applied in linear and nonlinear control systems.

2.3 Symbolic Computation in Control Systems

Several symbolic software packages exist that implement different theoretical results, developed within various mathematical languages, and allow to solve various control problems. (Polyakov, Ghanadan, & Blankenship, 1994) developed mathematica package called “CAS” for nonlinear and

adaptive control law synthesis and simulation. CAS provides functions for basic mathematical operations frequently encountered in analysis of nonlinear systems, implements algorithms for adaptive and approximate nonlinear control. (Ogunye, Penlidis, & Reilly, 1996) describes a collection of algorithms developed in a computer algebra package (MapleV) using polynomial matrix theory. The developed algorithms provide a medium in which polynomial matrix operations are carried out. Most importantly, these polynomial matrix procedures, enable the design and analysis of multivariable control systems using the algebraic or polynomial equation approach. (Abdalla, Wang, & McLauchlan, 1996) developed a Maple language procedure to solve continuous Lyapunov equations symbolically in conjunction with the Kronecker product. A comprehensive CACSD package SYMCON, for the design of multivariable discrete-time control systems using the polynomial equation approach has been described by (Ogunye, 1996). (Guay, McLellan, & Bacon, 1997) used the computer algebra program Maple to generate state and state feedback transformations using elements of the Lie Symmetry package and custom code where several dynamic feedback linearizable systems are used to illustrate the implementation. (Benyo, Palancz, Juhadz, & Varady, 1998) applied symbolic computation to design procedure for blood glucose control of diabetic patients under intensive care. To design of multivariable modal control based on nonlinear state-space model fully symbolically, MAPLE packages were employed. The model have been verified and tested in clinical environment. (Panjapornpon, Soroush, & Seider, 2006) presented a new software package that carries out symbolic manipulations to generate automatically analytical, model-based controllers and subsequently test the performance of the designed controller implemented on the process model. The software package has a user-friendly interface that was developed using Visual Basic and linked to MATHEMATICA using MathLink. Polynomial Control System is created by (Soylemez & Ustoglu, 2007) for Mathematica which presents new tools for the modeling, analysis, and design of linear control systems described by polynomial matrix equations or matrices with rational polynomial entries and PolyX (PolyX Ltd, 1998) which is a package for

systems, signals and control analysis and design based on advanced polynomial methods and written for MATLAB. However, both deal only with linear systems. In addition, there exist a number of smaller packages designed for specific problems, mostly in Maple or MATLAB software. OreModules, a software package, offers symbolic tools to investigate the structural properties of multidimensional linear systems over Ore algebras (Chyzak, Quadrat, & Robertz, 2007). NelinSys designed for analysis, design and simulation of nonlinear control systems (Ondera, 2005). (Antritter & Middeke, 2011) developed Maple-based package implementing concepts based on flatness theory. (Abramov, Le, & Li, 2005) developed OreTools which is a general-purpose package including theory of Ore algebras. DAISY is a software based on the methods of differential algebra for identifiability of systems (Bellu, Saccomani, Audoly, & D'Angiò, 2007).

CHAPTER 3: MATLAB

3.1 Introduction

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by “Math Works”, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python. Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the “MuPAD” symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems. Here, we will give an overview of MATLAB and its symbolic toolbox.

3.2 MATLAB Overview

MATLAB is a popular mathematical programming language based on matrix computation in the floating point domain. MATLAB also offers certain Maple functionality through the “Symbolic Toolbox”. It offers an interactive command-line interface, and provides a variety of plotting and graphical functionality in pop-up windows. MATLAB consists of the main program, with a variety of Toolboxes associated with it that may be used for different applications. It is probably the most common choice for engineers working with control systems. The MATLAB Control System Toolbox allows for the creation and manipulation of Linear Time Invariant (LTI) objects, a MATLAB data type. These objects may be instantiated using a transfer function format or a state space model. LTI objects may also be created by pole placement methods or even through graphical pole placement. Model conversion and extraction is possible. LTI objects may be added, multiplied, and concatenated easily with overloaded arithmetic operations. Graphical analysis is available, such as the Bode diagram, the Nyquist diagrams, and the Nichols chart. Time domain responses are available for

impulse or step inputs. MATLAB's Robust Control Toolbox is an additional collection of functions for the design of control systems. This Toolbox contains functions for so-called H_2 and H_∞ control, singular value analysis, and Riccati equation tools. A software tool for MATLAB called Simulink is a graphical tool for the construction and analysis of control systems. This tool allows the user to draw out the desired system using a GUI, and then perform simulations of system response. The systems may be continuous or discrete with respect to time, and may be linear or nonlinear. Third party software for MATLAB is abundant. For example, Shinnars [Stanley M. Shinnars. Modern Control System Theory and Design. John Wiley Sons, Inc., 1998.] provides freeware for use by readers of his textbook, called the Modern Control System Theory and Design Toolbox. This is a general-purpose package of control functionality that is based around the textbook. It has a full set of graphical routines as well as some analytical routines, including a small set of polynomial arithmetic functionality. Other third party software for MATLAB may be more specialized. For example, the Submarine Control Toolbox allows the user to design and simulate a virtual submarine. Typical design tools are present, including feedback control and eigenvalue assignment. The resulting design may be previewed in an impressive 3D graphical interface.

The MATLAB application is built around the MATLAB language, and most use of MATLAB involves typing MATLAB code into the Command Window (as an interactive mathematical shell), or executing text files containing MATLAB code, including scripts and/or functions. Variables are defined using the assignment operator, `=`. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function. MATLAB supports elements of lambda calculus by introducing function handles, or function references, which are implemented either in `.m` files or anonymous /nested functions.

3.3 Symbolic Toolbox

The Symbolic Math Toolboxes incorporate symbolic computation into the numeric environment of MATLAB. These toolboxes supplement MATLAB numeric and graphical facilities with several other types of mathematical computations. The computational engine underlying the toolboxes is the kernel of Maple, a system developed primarily at the University of Waterloo, Canada and, more recently, at the Eidgenössische Technische Hochschule, Zürich, Switzerland. Maple is marketed and supported by Waterloo Maple, Inc.

The basic Symbolic Math Toolbox is a collection of more than 100 MATLAB functions that provide access to the Maple kernel using a syntax and style that is a natural extension of the MATLAB language. The basic toolbox also allows you to access functions in the Maple linear algebra package. The Extended Symbolic Math Toolbox augments this functionality to include access to all non-graphics Maple packages, Maple programming features, and user-defined procedures. With both toolboxes, you can write your own M-files to access Maple functions and the Maple workspace.

The symbolic toolbox is a bit difficult to use but it is of great utility in applications in which symbolic expressions are necessary for reasons of accuracy in calculations. The toolbox simply calls the MAPLE kernel with whatever symbolic expressions you have declared, and then returns a (usually symbolic) expression back to MATLAB. It is important to remember that MAPLE is *not* a numeric engine, which means that there are certain things it doesn't let you do that MATLAB can do. Rather, it is useful as a supplement to provide functions which MATLAB, as a numerical engine, has difficulty with.

The symbolic math toolbox takes some time to initialize, so if nothing happens for a few seconds after you declare your first symbolic variable of the session, it doesn't mean you did anything wrong.

The MATLAB student version comes with a copy of the symbolic math toolbox.

3.3.1 What MATLAB Symbolic Toolbox can do?

You can declare a single symbolic variable using the 'sym' function as follows.

```
>> a = sym('a1')
a = a1
```

You can create arrays of symbolic expressions like everything else:

```
>> a1 = sym('a1');
>> a2 = sym('a2');
>> a = [a1, a2]
a = [ a1,  a2]
```

Symbolic variables can also be declared many at a time using the 'syms' function. By default, the symbolic variables created have the same names as the arguments of the 'syms' function. The following creates three symbolic variables, a b and c.

```
>> syms a b c
>> a
a = a
```

You can create functions of symbolic variables, not just the variables themselves. This is probably the most intuitive way to do it:

```
>> syms a b c %declare variables
>> f = a + b + c
ans = a + b + c
```

If you do it this way, you can then subsequently perform substitutions, differentiations, and so on with respect to any one of these variables.

The symbolic math toolbox is able to solve an algebraic expression for any variable, provided that it is mathematically possible to do so. It can also solve both single equations and algebraic systems. MATLAB uses the 'solve' function to solve an algebraic equation. The syntax is solve(f, var) where f is the function you wish to solve and var is the variable to solve for. If f is a function of a single variable you will get a *number*, while if it is multiple variables you will get a symbolic expression.

First, let us say we want to solve the quadratic equation $x^2 = 16$ for x. The solutions are $x = -4$ and $x = 4$. To do this, you can put the function into 'solve' directly, or you can define a function in terms of x to solve and pass that into the 'solve' function. The first method is rather intuitive:

```
>> solve('x^2 = 16', x)
ans = -4
      4
>> solve(x^2 - 16, x)
ans = -4
      4
```

Either of these two syntax works. The first must be in quotes or you get an 'invalid assignment' error. In the second, x must be defined as a symbolic variable beforehand or you get an 'undefined variable' error.

For the second method you assign a dummy variable to the equation you want to solve like this:

```
>> syms x
>> y = x^2 - 16;
>> solve(y, x);
```

Note that since MATLAB assumes that $y = 0$ when you're solving the equation, you must subtract 16 from both sides to put the equation into normal form.

In addition to all mentioned above, MATLAB symbolic toolbox can do many analytical problem solving like integration, differentiation, systems of equations, differential equations, ... etc. What have been mentioned here should be enough for the reader to use the NLC package. For more information, you may refer to MATLAB manual or documentations.

CHAPTER 4: INPUT-OUTPUT EXACT LINEARIZATION

4.1 Introduction

A classical way to control nonlinear systems is to compute a linear controller using the first-order approximation of the system dynamics around the origin $x = 0$, which gives a local linear approximation of the system. A non-approximating linearization called input-output exact feedback linearization is discussed here. Feedback linearization is a common approach used in controlling nonlinear systems. The approach involves coming up with a transformation of the nonlinear system into an equivalent linear system through a change of variables and a suitable control input. The goal is to develop a control input u that renders a linear input–output map between the new input v and the output. An outer-loop control strategy for the resulting linear control system can then be applied. In this chapter, the methodology of exact input-output linearization will be explained.

4.2 Affine and Non-affine Nonlinear Systems

Generally, the nonlinear systems are described in the following state space form:

$$\begin{aligned}\dot{x} &= f(x, u) \\ y &= h(x)\end{aligned}\tag{1}$$

Where x is the n -dimensional state space variable vector, u the m -dimensional input vector, y the p -dimensional output vector, f the state vector function and h the output vector function. When a system belongs to this class of systems, the system is called a non-affine nonlinear system or a system not affine in the input. The class of systems most writers refer to is a special case of nonlinear systems: affine nonlinear systems or systems affine in the input. Systems belong to this class when:

$$\frac{\partial f(x, u)}{\partial u} = g(x)\tag{2}$$

If this is true (and g is not a function of u) the system is an affine nonlinear system (system affine in the input) and can be written in the following way:

$$\dot{x} = f(x) + g(x)u \quad (3)$$

$$y = h(x)$$

With g the input matrix function.

It is however always possible to change a non-affine system into an affine system so that the theories applied to affine systems suffices also in cases where we are dealing with non-affine systems. The change into an affine system is done by extending the state x with the input u (Henson & Seborg, 1997).

$$\bar{x} = \begin{bmatrix} x \\ u \end{bmatrix} \quad (4)$$

and by the definition of the new input:

$$\bar{u} = \dot{u} \quad (5)$$

The non-affine system can now be written in the following way:

$$\dot{x} = f(x) + G(x, u) = f(x) + G(\bar{x}) \quad (6)$$

and the extended affine system

$$\begin{bmatrix} \dot{x} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} f(x) \\ 0 \end{bmatrix} + \begin{bmatrix} G(\bar{x}) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \bar{u} \quad (7)$$

This is the affine form of the same system with new f and g . It can be written as follows:

$$\dot{\bar{x}} = \bar{f}(\bar{x}) + \bar{g}(\bar{x})\bar{u} \quad (8)$$

The inputs (\bar{u}) adds an integrator to the system and this has to be handled with care. The extra integrator gives the system a phase lag and this can compromise the stability of the system.

4.3 The Relative Degree

For linear systems, the input-output property of relative degree corresponds to the difference between the number of poles and the number of zeros of the system. For nonlinear systems, the relative degree of System (3) corresponds to the number of times the output $y = h(x)$ has to be differentiated with respect to time before the input u appears explicitly in the resulting equations.

The objective of the method of input-output exact linearization is to control a nonlinear system given in the form (3) by turning it into a linear and controllable one i.e. such that can be described by linear state-space equations:

$$\dot{y} = v \tag{9}$$

Consider the state space system in (3), in case of single input single output (SISO), the time derivative of y is then given by:

$$\begin{aligned} \dot{y} &= \frac{\partial h}{\partial x} \dot{x} \\ &= \frac{\partial h}{\partial x} (f(x) + g(x)u) \\ &= L_f h(x) + L_g h(x)u \end{aligned} \tag{10}$$

$L_f h(x)$ is called the lie derivative along the vector field f and it is defined as follows:

$$\begin{aligned} L_f h(x) &= \frac{\partial h}{\partial(x)} f(x) \\ L_g L_f h(x) &= \frac{\partial(L_f h)}{\partial(x)} g(x) \end{aligned} \tag{11}$$

$$L_f^i h(x) = L_f L_f^{i-1} h(x), \quad \text{where } L_f^0 h(x) = h(x)$$

In above equation (10), if $L_g h(x) \neq 0$, then the control law is given by:

$$u = \frac{1}{L_g h(x)} (-L_f h(x) + v) \quad (12)$$

yielding $\dot{y} = v$. Otherwise, (that is $L_g h(x) = 0$), differentiate once more:

$$\dot{y} = \frac{\partial L_f h}{\partial x} (f(x) + g(x)u) = L_f^2 h(x) + L_g L_f h(x)u \quad (13)$$

if $L_g L_f h(x) \neq 0$, then the control law is given by:

$$u = \frac{1}{L_g L_f h(x)} (-L_f^2 h(x) + v) \quad (14)$$

yielding $\ddot{y} = v$ and so on. So, basically we differentiate until u appears. Number of derivatives required for u to appear is called the relative degree r which is defined as follows (Isidori, 2013; Khalil, 2001; Slotine, Li, & others, 1991).

Definition 4.1: There exists an integer $r \leq n$ such that $L_g L_f^i h(x) = 0$ for all $i \in \{1, \dots, r-2\}$ and $L_g L_f^{r-1} h(x) \neq 0$ that is called the relative degree.

The notion of the relative degree is an important property of a system. It is important because of the fact that most theory coming up in this chapter is applicable only if the system has a well-defined relative degree. The term "well-defined" will be made clear further in this section. The relative degree r denotes exactly the number of times the output has to be differentiated until the input explicitly appears in the output.

For MIMO systems, with m inputs and p outputs ($m \geq p$), we can expand the definition of the relative degree. The relative degree now consists of a vector, the vector relative degree, with elements which denote the relative degree of each output channel. So the vector relative degree consists of p elements, $[r_1, r_2, \dots, r_p]$, where r_i denotes the number of times output y_i has to be differentiated until at least one of the inputs appear in that output y_i . The outputs derivatives of MIMO systems can be written as follows:

$$\frac{d^k y}{dt^k} = L_f^k h_i(x), \quad k = 1, \dots, r_{i-1} \quad (15)$$

$$\frac{d^{r_i} y_i}{dt^{r_i}} = L_f^{r_i} h_i(x) + \sum_{j=1}^m L_{g_j} L_f^{r_i-1} h_i(x) u_j$$

If a system output y_i does not have a relative order, this means that y_i and all its derivatives are not explicitly dependent on u ; consequently, y_i is not affected by u . In every well-formulated control problem, all outputs y_i must possess a relative order. Otherwise, the system will not be output controllable. The definition of the relative degree for MIMO systems can be written as follows (Isidori, 2013):

$$L_g L_f^k h_i(x) = 0 \quad \forall k < r_i - 1, \quad 1 \leq i \leq p \text{ and } 1 \leq j \leq m \quad (16)$$

Definition 4.2: Consider a system of the form of equation (1) and assume that each output y_i has a relative degree r_i , the matrix:

$$E(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1 & \cdots & L_{g_m} L_f^{r_1-1} h_1 \\ \vdots & \cdots & \vdots \\ L_{g_1} L_f^{r_p-1} h_p & \cdots & L_{g_m} L_f^{r_p-1} h_p \end{bmatrix} \quad (17)$$

is called decoupling matrix or characteristic matrix. In this matrix, immediately it can be seen which input(s) will affect which output(s). Row i relate to output i and column j related to input j . This vector relative degree is not well defined in the following cases:

1. As the rank of E is not equal to p because of a zero-row. This means that output i is not affected by any input at all.
2. As each output channel apart has a well-defined relative degree but the rank of E is less than p due to zero-columns which means that one of the input channels does always appear later than at least one of the other input channels, in all outputs.

In some cases where the rank of decoupling matrix is less than number of outputs (p), it is possible to achieve a well-defined relative degree via dynamic extension which is going to be explained in the coming section.

4.4 Dynamic Extension

If the relative degree for the system is not defined because one or more input(s) appear always later than other(s), then the decoupling matrix E does not have a rank equal to the number of output(s). By adding the early input to the state, the appearance of this input is "delayed" to higher orders of the outputs and it is then possible that the late input shows up while determining the relative degree. It is then possible that the matrix E has a rank equal to the number of output(s) and the relative degree becomes defined (Isidori, 2013). Following simple example will make this clearer.

Example 4.1: consider a MIMO system with following state space:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \cos x_3 & 0 \\ \sin x_3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Applying formula given by (17) gives the following decoupling matrix $E = \begin{bmatrix} \cos x_3 & 0 \\ \sin x_3 & 0 \end{bmatrix}$ which is clear has a rank 1 less than the number of outputs 2 and thus relative degree is not well-defined. Let's define a new input $\dot{u}_1 = \mu$ which will extend the original system to the following:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} u_1 \cos x_3 \\ u_1 \sin x_3 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \mu \\ u_2 \end{bmatrix}$$

Now decoupling matrix becomes $E = \begin{bmatrix} \cos x_3 & -u_1 \sin x_3 \\ \sin x_3 & u_1 \cos x_3 \end{bmatrix}$ which has a rank of 2 equals to the number of outputs and thus relative degree is well-defined.

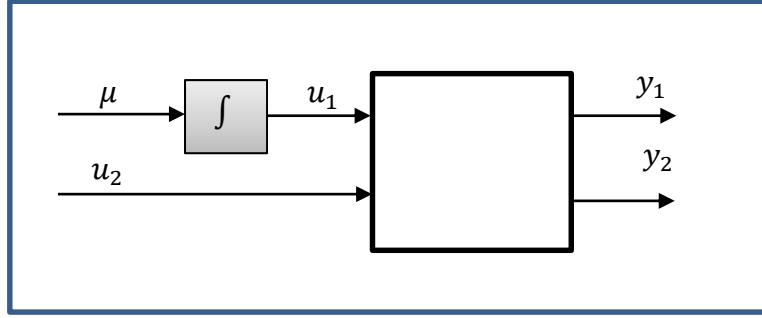


Figure 1 Demonstration of Dynamic Extension

What we have done in previous example is that the affecting input u_1 has been delayed by adding an integrator which make the higher output derivative show non-affecting input u_2 (see figure 4.1).

4.5 State Transformation

As we have mentioned earlier, the idea of input-output exact linearization is to control a nonlinear system given in the form (2) by turning it into a linear and controllable one ($\dot{y}_i = v_i$). So, the linearized states are actually the output derivatives and can be written as follows for SISO systems:

$$z_1 = y, z_2 = \dot{y}, \dots, z_r = y^{(r-1)} \rightarrow z_i(x) = L_f^{i-1} h(x), \quad i = 1, \dots, r \quad (18)$$

For MIMO systems, the state transformation takes the following form (Henson & Seborg, 1997; Isidori, 2013):

$$z(x) = \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ L_f^{r_1} h_1(x) \\ \vdots \\ h_m \\ \vdots \\ L_f^{r_m-1} h_m(x) \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_1^{r_1-1} \\ \vdots \\ y_m \\ \vdots \\ y_m^{m-1} \end{bmatrix} \quad (19)$$

When the relative degree is equal to the number of states then the states transformation is already complete. If $r < n$, then it is possible to find an additional $n - r$ functions so that a local coordinate transformation is reached. When input-output linearization is applied to a system with $r < n$ then $n - r$ system equations (internal dynamics) remain which are not input-output linearized. In order for

the controller to be useful in practice, these equations have to define a stable system. A way to study the stability of these equations is by looking at the zero-dynamics. The zero-dynamics are defined to be the internal dynamics of a system when the system output(s) are kept at zero by the input(s). A nonlinear system whose zero-dynamics is stable is called a “minimum phase” system. Especially for the analysis of the tracking dynamics of a system it can be very useful if the input(s) don't appear in these last equations. Then there are no restrictions on the input(s) for the stability of the zero-dynamics. Sometimes, it is possible to achieve full order relative degree ($r = n$) using dynamic extension. It is also very important to know that when having a full order relative degree then it is not necessary to check stability of internal dynamics. So, the case of $r < n$ is not going to be considered in our MATLAB package, instead will try to achieve full order relative degree using dynamic extension.

4.6 Feedback Linearizing Control Law

If a system has a relative degree (r_1, \dots, r_m) and the total relative degree $(r_1 + r_2 + \dots + r_m)$ equals to n , then this system can be rendered linear by means of a feedback and a transformation of states. One can at least obtain a system whose input-output behavior is linear. The latter condition is not necessary, but we will commit to this condition (total relative degree equals system order, n) since the case of total relative degree less than n is not considered here.

The control law can be derived from the output(s) derivative(s). Recall the definition of relative degree in section 4.2 where we mentioned that it is the number of output derivatives (r) required until input appears. The r^{th} derivative of SISO system can be written as follows:

$$y^{(r)} = L_f^r h(x) + L_g L_f^{r-1} h(x) u \quad (20)$$

the feedback linearizing control law yielding to $\dot{y} = v$, is given by:

$$u = \frac{1}{L_g L_f h(x)} (-L_f^2 h(x) + v) \quad (21)$$

Similarly, for MIMO systems the output derivatives can be written as follows (Henson & Seborg, 1997; Isidori, 2013; Khalil, 2001):

$$\begin{bmatrix} y^{(r_1)} \\ \vdots \\ y^{(r_p)} \end{bmatrix} = \underbrace{\begin{bmatrix} L_f^{(r_1)} h_1(x) \\ \vdots \\ L_f^{(r_p)} h_p(x) \end{bmatrix}}_{b(x)} + \underbrace{\begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1 & \cdots & L_{g_m} L_f^{r_1-1} h_1 \\ \vdots & \cdots & \vdots \\ L_{g_1} L_f^{r_p-1} h_p & \cdots & L_{g_m} L_f^{r_p-1} h_p \end{bmatrix}}_{E(x)} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}}_u \quad (22)$$

Under the condition that rank of E equals to p , the control law is given by:

$$u = E(x)^{-1} \left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} - b(x) \right) \quad (23)$$

which gives following chains of integrators:

$$y_1^{(r_1)} = v_1, y_2^{(r_2)} = v_2, \dots, y_p^{(r_p)} = v_p \quad (24)$$

Exact linearization techniques can be interpreted as shown in below figure 2. In this figure $z(x)$ is

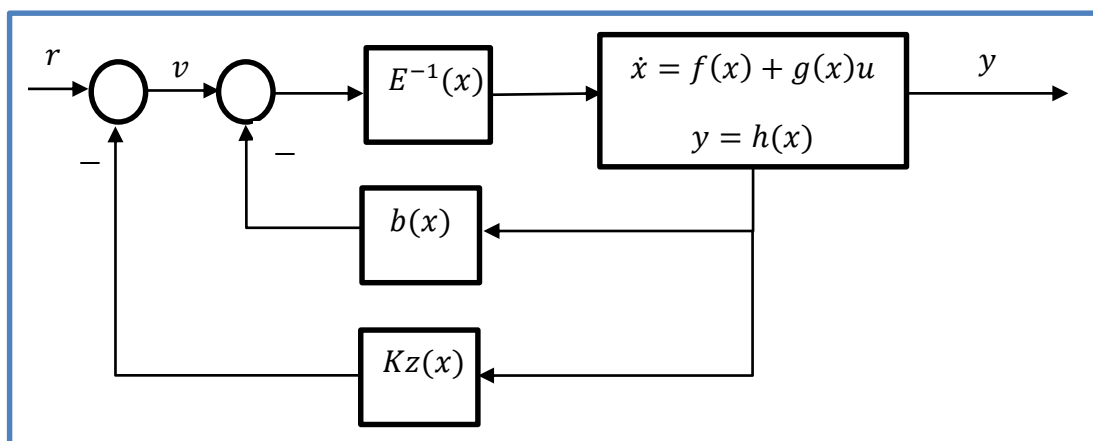


Figure 2 Feedback linearization interpretation

the state transformation given in equation (19) and K is the linear controller gain.

4.7 Exact linearization for non-square systems

For a non-square MIMO system represented by (3), E is a $p \times m$ matrix, E^{-1} is not defined. Thus, Eq. (23) cannot be used to define new inputs required for I/O linearization. Instead of inverse calculation, in non-square systems we calculate the Moore-Penrose pseudo-inverse of the decoupling matrix E denoted by $E^\#$ and its existence is guaranteed by the assumption that it has full row rank. So, the I/O linearization feedback law can be written as follows:

$$u = E^\#(x) \left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} - b(x) \right) \quad (25)$$

The calculation of pseudo-inverse requires solution to the following optimization problem. Consider the solution to a set of algebraic equations in which the number unknowns is greater than the number of equations, i.e. of the form $Ax = b$. There are infinite solutions to the above set of equations. However, the solution that minimizes a cost function defined by $J = \|Wx\|_2^2$, is unique and can be obtained by solving the following problem:

$$\min \|Wx\|_2^2, \quad s. t. Ax = b \quad (26)$$

where W is the weight of the cost function. The solution to the above optimization problem can be interpreted as finding a pseudo-inverse of A . In other words, the solution that is obtained can be written as $x = A^\#b$, where $A^\#$ is the Moore-Penrose pseudo-inverse of A . For more information, the reader may refer to literatures (Noble, 1976; Kolavennu et al., 2001). The Moore-Penrose pseudo-inverse can be calculated in MATLAB using built in function "*pinv*".

CHAPTER 5: The NLC Package

5.1 Introduction

The NLC package is a collection of functions and subroutines that is designed to perform exact linearization symbolically. Each function is saved in a separate "m" file where all files have to be saved in the same directory. The NLC can be started by running `nlc_main.m` file or simply by typing `nlc_main` in MATLAB command window. NLC is a text based package with a self-guiding user interface. It has been designed to overcome limitations in previous work and minimize the amount of user-input data and give a very clear documentation on the executed results. In this chapter, each function will be explained and algorithms will be presented. In last section of this chapter, the package will be tested using several examples demonstrating the package features.

5.2 Package Overview and Features

NLC starts with a list of models to be selected to perform the exact linearization analysis. There is an option to edit a user-defined model. Once this option is selected, m file editor will be opened and instructions are giving on how to edit the file and write differential equations. MATLAB command will be on hold waiting for the user response to continue. Once the model is entered, user is instructed to save the file and return to MATLAB command window and hit any key to continue. The first thing the program will do is to check the affine system. If the system is affine, it will display a message and continue the analysis. Otherwise, it will convert it to an affine system (refer to 4.1 for more details on how to convert non-affine into affine). Then, the program will display the states, input(s), $f(x)$, $g(x)$, and $h(x)$. The next step is to calculate the relative degree vector, decoupling matrix E , and output derivatives b . The result of the analysis will be shown. Here, there could be three different situations:

1. Rank of $E < p$ where it will give a warning that relative degree is not well defined and will give an option to run dynamic extension to achieve well-defined relative degree.
2. Rank of $E = p$ and the sum of relative degree vector (r-total) is less than system order, n then it will show a warning that the system may have internal dynamics unstable and will give an option to run dynamic extension to achieve full order relative degree.
3. Rank of $E = p$ and *sum of relative degree vector* = n , then the system is having full order and well-defined relative degree, therefore the control law and state transformation will be calculated and the state space matrices of the linearized system will be shown.

For more information on the above three cases of relative degree, please refer to 4.2.

Once the control law is calculated and shown, the program will give an option to generate code for which it can be used for simulation. This code can be used in SIMULINK MATLAB Function block. This block is designed to accept MATLAB command that can be used inside Simulink. All what user have to do next is to connect the outputs of the block to integrator(s), except those for $h(x)$ where they are usually connected to display or scope function block, and connect output(s) of integrator(s) to the input(s) of the MATLAB function block, except those for desired outputs (reference inputs). The entire program execution flowchart is presented in Appendix A and the program source code is available in Appendix B. Each subroutine will be explained in detail with algorithms in the following sections. Several examples will be shown and demonstrated. Below is a summary of the NLC package features that distinguish it from others:

1. The system is to be entered as a set of differential equations and algebraic equations (if any) which mean the computation effort by user is minimized/ eliminated.
2. It Supports SISO and MIMO systems without any intervention from user. In other words, the system is dealing with both without asking the user to identify whether it is SISO or MIMO.

3. It accepts non-square systems with condition $m \geq p$ (number of inputs is greater than or equal to number of outputs).
4. It converts non-affine into affine system since the theory is based on affine systems.
5. It performs dynamic extension to convert into affine and possibly get full order and well-defined relative degree.
6. It immediately shows the state space matrices for the linearized system where it can be easily used to analyze and design linear controller.
7. It generates codes for use with MATLAB Function SIMULINK block. LQI controller will be included in the code (refer to section 5.8 for more details). The user will be asked to give states and input weights.
8. It offers self-guidance and easy text based interface. All results will be show with useful comments and demonstrations.

5.3 Converting to affine, "nlc_affine" subroutine

This subroutine tests the differential equations $f(x, u)$ to see if the system is affine in input. If the system is not affine, it will change the system to affine by introducing new state as integration of non-affine input. The inputs are: differential equations vector, $f(x, u)$, states of the system, and input(s), u . Outputs are: state functions vector, $f(x)$, input(s) vector, $g(x)$, new differential equations vector, $\bar{f}(x)$, new states, \bar{x} , and new inputs, \bar{u} . The last three outputs will be same as inputs in case the system is affine already. The algorithm is shown below.

1. Compute partial derivatives of $f(x, u)$ with respect to inputs and assign result to $g(x) \rightarrow$

$$g(x) := \frac{\partial f(x, u)}{\partial u}$$
2. Let $gvar$ be a vector contains the list of all variables in $g(x)$
3. If $gvar$ is not containing any elements equals to any elements in u , then the system is affine else the system is not affine

4. If the system is affine, go to 8 else go to 5
5. Define new state and input $\rightarrow \bar{x} := \begin{bmatrix} x \\ u \end{bmatrix}, \bar{u} := \dot{u}$
6. Compute $\bar{f}(x, u)$ by substituting \bar{x} and \bar{u} into $f(x, u)$
7. Go to 1.
8. Compute $f(x) := f(x, u) - g(x) * u$
9. Stop

Steps 5 to 7 is executed by a different subroutine called "nlc_MakeAffine". So, false condition of step 4 is executed by calling "nlc_MakeAffine" and step 7 is executed by calling original subroutine "nlc_affine".

5.4 Relative Degree Calculation, "nlc_RelDeg" subroutine

This subroutine computes the relative degree vector. Inputs are $f(x), g(x), h(x)$, and u . Outputs are relative degree vector and total relative degree (in case of SISO both will have the same value). This subroutine is calling "nlc_lieDerivative" which calculates lie derivate. Recall the definition of lie derivative giving by (11), the lei derivative subroutine is using that definition to calculate the first order lie derivative. The higher order lie derivative is calculated in the original subroutine "nlc_RelDeg" recursively. The algorithm is shown below.

1. $i := 0, m := \text{number of inputs}, p := \text{number of ouptus}, k := 1$
2. $i := i + 1$
3. If $k = 1$, then $Lfh := h_i(x)$, else go to 8
4. $j := 0$
5. $j := j + 1$
6. $LgLfh_j := L_{g_j}h_i(x)$
7. If $j \leq m$, then go to 5, else go to 8

8. $r_i := k$
9. $Lfh_{new} := L_f(Lfh)$
10. $j := 0$
11. $j := j + 1$
12. $LgLfh_j := L_{g_j}(Lfh_{new})$
13. $Lfh := Lfh_{new}$
14. If $j \leq m$, then go to 11, else go to 15
15. $z := 0$
16. $z := z + 1$
17. If $LgLfh_z \neq 0$, then go to 20, else $k := k + 1$
18. If $z \leq m$, then go to 16, else go to 19
19. Go to 3
20. If $i \leq p$, then go to 2, else go to 21
21. $r_{vector} := r_1, \dots, r_i$
22. $r_{total} := r_1 + \dots + r_i$
23. Stop

5.5 Decoupling and output derivatives matrices, "nlc_decoupling" subroutine

This subroutine computes decoupling (characteristic) matrix, E and output derivatives vector, b . In fact it is applying the following formula (refer to section 4.3 for more details on derivation of these formulas):

$$E(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1 & \cdots & L_{g_m} L_f^{r_1-1} h_1 \\ \vdots & \cdots & \vdots \\ L_{g_1} L_f^{r_p-1} h_p & \cdots & L_{g_m} L_f^{r_p-1} h_p \end{bmatrix}, b(x) = \begin{bmatrix} L_f^{(r_1)} h_1(x) \\ \vdots \\ L_f^{(r_p)} h_p(x) \end{bmatrix}$$

The inputs to this subroutine are $f(x)$, $g(x)$, $h(x)$, x , u , and r_{vector} while the outputs are $E(x)$ and $b(x)$. This subroutine calls "nlc_leiderivative" subroutine as well. The algorithm is presented in the following paragraph.

1. $i := 0$, $m :=$ number of inputs, $p :=$ number of outputs
2. $i := i + 1$
3. $k := 0$
4. $k := k + 1$
5. If $k = 1$, then $Lfh := h_i(x)$, else go to 10
6. $j := 0$
7. $j := j + 1$
8. $LgLfh_j := L_{g_j}h_i(x)$
9. If $j \leq m$, then go to 7, else go to 10
10. $Lfh_{new} := L_f(Lfh)$
11. $j := 0$
12. $j := j + 1$
13. $LgLfh_j := L_{g_j}(Lfh_{new})$
14. $Lfh := Lfh_{new}$
15. If $j \leq m$, then go to 12, else go to 5
16. If $k \leq r_i$, then go to 4, else go to 16
17. $b_i := L_f(Lfh)$
18. i^{th} row of $E := L_g(Lfh)$
19. If $i \leq p$, then go to 2, else go to 20
20. Stop

5.6 Dynamic Extension, "nlc_dynamicExt" subroutine

This subroutine perform dynamic extension in a very simple idea. It is looking at the decoupling matrix, E and identify the index of the column having minimum zeros. If more than one column is having the same number of zeros it will select the first one. This column index correspond to the most input having effect on all inputs and this input is selected to be delayed by adding an integrator to it which means defining new state and new input (refer to section 4.4 for more details). The program will display the input number selected in every iteration and recalculate relative degree and decoupling matrix and displays them. The program will terminate in either following conditions:

1. The relative degree becomes well-defined and equals to the number of states. Here, the algorithm succeeds.
2. The iteration number reaches maximum number selected by the user (normally the maximum iteration number is selected to be equals to number of states). Here the algorithm fails.

The inputs to this subroutine is $f(x), g(x), E, x$, and u while the outputs is new set of functions $\bar{f}(x), \bar{g}(x), \bar{E}, \bar{x}$, and \bar{u} . In case the algorithm succeeds it will continue the analysis by executing the next subroutines namely, the state transformation and feedback linearizing controller which will be described in the coming sections. Below, the algorithm is presented.

1. $p :=$ number of outputs
2. Prompt user to choose value of the maximum iteration and assign it to i_{maximum}
3. $i := 0$
4. $i := i + 1$
5. $j := 0$
6. $j := j + 1$
7. $m :=$ number of inputs, $n :=$ number of states
8. Find non-zero elements in column j of E and store the total number in element j of the vector E_{nonzero}
9. If $j \leq m$, then go to 6, else go to 10
10. $u_i :=$ maximum of E_{nonzero}
11. Define new state and input $\rightarrow x_n := \begin{bmatrix} x \\ u_{u_i} \end{bmatrix}, u_n := u_{u_i}$
12. Compute $f_n(x), g_n(x)$ by substituting x_n and u_n into $f(x), g(x)$
13. Compute $r_{\text{vector}}, r_{\text{total}}$ by executing "nlc_RelDeg" subroutine
14. Compute E_n by executing "nlc_decoupling" subroutine
15. If rank of $E_n = p$ and $r_{\text{total}} = n$, then go to 16, else go to 18

16. $x := x_n, f := f_n, u := u_n, g := g_n$
17. If $i \leq i_{maximum}$ then, go to 4, else go to 18
18. Stop

5.7 State Transformation, "nlc_StatTrans" subroutine

This subroutine is direct application of the formula (19) where the derivative of each output is calculated with respect to its relative degree. The inputs are $f(x), h(x), r_{vector}$ and outputs are $Z = z_1, z_2, \dots, z_n$. In the next paragraph, algorithm is shown.

1. $p :=$ number of outputs
2. $i := 0$
3. $i := i + 1$
4. $k := 0$
5. $k := k + 1$
6. If $k = 1$, then $Lfh := h_i(x)$, else go to 7
7. $Lfh_{new} := L_f(Lfh)$
8. $Lfh := Lfh_{new}$
9. $z_k := Lfh$
10. If $k \leq r_i$, then go to 5, else go to 11
11. If $i = 1$, then $Z := z$, else $Z := (Z, z)$
12. If $i \leq p$, then go to 3, else go to 13
13. Stop

5.8 Feedback linearizing controller, "nlc_FBL" subroutine

Recall the control law for square system, equation (23):

$$u = E(x)^{-1} \left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} - b(x) \right)$$

and for non-square system, equation (25):

$$u = E^\#(x) \left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} - b(x) \right), \text{ where } E^\# \text{ is Moore-Penrose pseudo-inverse. The "nlc_FBL" subroutine}$$

checks the size of the system and compute the control law accordingly. The inputs to this subroutine are as follows: $E(x)$, and $b(x)$ which are computed in "nlc_decoupling" subroutine. The output is the control law. Algorithm is presented below.

1. $m :=$ number of inputs, $p :=$ number of outputs
2. If $p = m$, then go to 3, else go to 7
3. Define $V = v_1, \dots, v_p$
4. Compute inverse of E , $E_{inv} := E^{-1}$
5. $u := E_{inv} * (V - b)$
6. Go to 9
7. Compute Moore-Penrose pseudo-inverse of E , $E_{pinv} := E^\#$
8. $u := E_{pinv} * (V - b)$
9. Stop

5.9 Simulation code generation, "nlc_WriteToFile" subroutine

One of the powerful tools of MATLAB is the SIMULINK. It is a block diagram environment for multi-domain simulation and Model-Based Design. It supports simulation, automatic code

generation, and continuous test and verification of embedded systems. In SIMULINK, there is a block which includes MATLAB code in models that generate embeddable C code. With a MATLAB Function block, you can write a MATLAB function for use in a SIMULINK model. The MATLAB function you create executes for simulation and generates code for a Simulink Coder target. The main function of this subroutine is to generate MATLAB code where it can be used in MATLAB Function block. This code contains differential equations of the system, state transformation, linear controller design, and feedback linearized controller. The code is saved in nlc_DiffEqFile.m where the user can open and copy the code to MATLAB function block. Following section contains an example explaining how to use this feature.

The linear control technique chosen is LQI (Linear Quadratic Integral control). Since the most application of the input-output exact linearization is the tracking control, LQI is selected because it is popular and powerful in this area. LQI is an optimal state-feedback control law for the tracking loop shown below. The state-feedback control is of the form $u = -K \begin{bmatrix} x \\ x_i \end{bmatrix}$ where x_i is the integrator output. This control law ensures that the output y tracks the reference command r . For MIMO systems, the number of integrators is equal to the dimension of the output y .

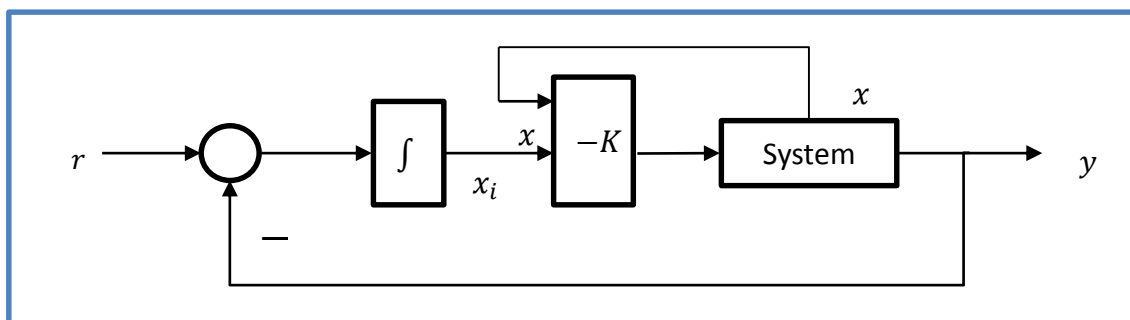


Figure 3 LQI interpretation

The built in MATLAB function "LQI" calculates the optimal gain matrix K given a state-space model system of the plant and weighting matrices Q, R , and N . In our case, the state space system is the equivalent linearized system and its state-space matrices is generated by subroutine "nlc_EquLinSys"). The control law $u = -K \begin{bmatrix} x \\ x_i \end{bmatrix} = -Kz$ minimizes the cost function:

$$J(u) = \int (z'Qz + u'Ru + zz'Nu)dt \quad (26)$$

Weighting matrix is neglected in our LQI design ($N = 0$). The error integration ($\int (r - y)dt$) is generated by introducing new states as shown below in equation (27).

$$\dot{x}_i = r - y \rightarrow x_i = \int (r - y)dt, \quad i = 1, \dots, p \quad (27)$$

5.10 Examples

In this section the NLC package will become clearer by representing different examples. Applications of simulation functions, changing to affine, dynamic extension, as well as some models that cannot be used in this package.

5.10.1 Contrived Model

The model used here is taken from (Isidori, 2013). The system has full order and well defined relative degree as it will be shown below. It is a MIMO system with 2 inputs and 2 outputs. The system dynamics is represented by:

$$\dot{x} = \begin{bmatrix} x_2 + x_2^2 + u_2 \\ x_3 - x_1x_4 + x_4x_5 \\ x_2x_4 + x_1x_5 - x_5 + u_1 \cos(x_1 - x_5) + u_2 \\ x_5 \\ x_2^2 + u_2 \end{bmatrix}, u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, h(x) = \begin{bmatrix} x_1 - x_5 \\ x_4 \end{bmatrix}$$

The system model is loaded already in the package, it will be shown in menu where the user can select it by typing the corresponding serial number. Below is the program output:

```
*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***
```

```
Below is a list of nonlinear models where you can select to apply
nonlinear control design. You can enter your own model as well.
```

```
For Help and more Information about this package typ 0 "zero"
```

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model
2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continuous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor
5. User Defined Model
6. General nonlinear model examples

Select a model by typing the corresponding number: 6

1. An example of Dynamic Extension Algorithm Failure (MIMO).
2. An example of Dynamic Extension Algorithm Success (MIMO).
3. Mechanical Arm (SISO).
4. The Contrived Model (MIMO).
5. Go back to previous menu.

Select a model by typing corresponding serial number: 4

System is affine!

States of the system are:

[x1, x2, x3, x4, x5]

f(x) =

$$x2^2 + x2$$

$$x3 - x1*x4 + x4*x5$$

$$-x_5^2 + x_1 x_5 + x_2 x_4$$

$$x_5$$

$$x_2^2$$

$$g(x) =$$

$$[\quad \quad \quad 0, 1]$$

$$[\quad \quad \quad 0, 0]$$

$$[\cos(x_1 - x_5), 1]$$

$$[\quad \quad \quad 0, 0]$$

$$[\quad \quad \quad 0, 1]$$

$$u =$$

$$u_1$$

$$u_2$$

$$h(x) =$$

$$x_1 - x_5$$

$$x_4$$

$$\text{Decoupling Matrix, } E =$$

$$[\cos(x_1 - x_5), 1]$$

$$[\quad \quad \quad 0, 1]$$

No. of States (n) is : 5

Total relative degree (r_total) is : 5

Relative degree vector (r_vector) is : [3 2]

Number of inputs (m) is : 2

Number of outputs (p) is : 2

Rank of decoupling matrix is : 2

Transformation Equations, Z =

$$z1 = x1 - x5$$

$$z2 = x2$$

$$z3 = x3 - x1*x4 + x4*x5$$

$$z4 = x4$$

$$z5 = x5$$

Nonlinear feedback, U =

$$u1 = (v1 - x1*x5 - x2*x4 + x5*(x1 - x5) + x4*(x2 + x2^2) - x2^2*x4 + x5^2)/\cos(x1 - x5) - (v2 - x2^2)/\cos(x1 - x5)$$

$$u2 = v2 - x2^2$$

Following is the linearized state space system:

A =

$$\begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

B =

```
0    0
0    0
1    0
0    0
0    1
```

C =

```
1    0    0    0    0
0    0    0    1    0
```

D =

```
0
```

Above A,B,C,D matrices can be used for linear controller desing. This package can prepare a code to be used in a simulink block "MatlabFunction" where you can perform simulation more effectively. The code will have an emeded LQI controller. To prepare the code type "nlc_WriteToFile"

Elapsed time is 6.397189 seconds.

As seen above, it is giving a full analysis of the system. Starting from testing affine to computing the linearized feedback control law. The next step is to get the code for the simulation by typing "nlc_WriteToFile" as it is instructed.

nlc_WriteToFile

LQI controller design....

please specify the states weight, q =: 50000

please specify the inputs gain, r =: 0.01

The code will be shown below, select and copy then past it in MatlabFunction block

which is in the Simulink file. Once you copy press any key then the Simulink file

will be opened. You can save the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired outputs. After running the simulink

you can plot the results using the command "nlc_PlotSimulation"

Press any key when your are ready...

%%%%%%%%%% Begining of the code %%%%%%%%%%

```
function [dx1,dx2,dx3,dx4,dx5,dx6,dx7,h1,h2] =  
nlc_DiffEqFile(x1,x2,x3,x4,x5,x6,x7,yd1,yd2)
```

```
z = [ x1 - x5;x2;x3 - x1*x4 + x4*x5;x4;x5;x6;x7 ];
```

```
K = [ 2577.8981,367.9581,27.1278,-1.0828e-13,-2.9891e-17,-2236.068,-5.5614e-13;-1.3811e-  
14,3.1095e-14,-2.9891e-17,2302.9348,67.8666,1.4718e-12,-2236.068 ];
```

```
v = -K*z;
```

```
v1 = v(1);
```

```
v2 = v(2);
```

```
u1 = (v1 - x1*x5 - x2*x4 + x5*(x1 - x5) + x4*(x2 + x2^2) - x2^2*x4 + x5^2)/cos(x1 - x5) -  
(v2 - x2^2)/cos(x1 - x5);
```

```
u2 = v2 - x2^2;
```

```
dx1 = u2 + x2 + x2^2;
```

```
dx2 = x3 - x1*x4 + x4*x5;
```

```
dx3 = u2 + x1*x5 + x2*x4 - x5^2 + u1*cos(x1 - x5);
```

```
dx4 = x5;  
dx5 = u2 + x2^2;  
h1 = x1 - x5;  
h2 = x4;  
dx6 = yd1 - h1;  
dx7 = yd2 - h2;
```

```
%%%%%%%%% End of the code %%%%%%%%%
```

Copy above code and past it in MatlabFunction block. You can save

the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired outputs. After running the simulink

you can plot the results using the command "nlc_PlotSimulation"

Press any key when your are ready...

This may take while, be patient please....

Below is the Simulink blocks where the generated codes is used in MATLAB Function block. The plot of the simulated outputs and tracking reference is show as well.

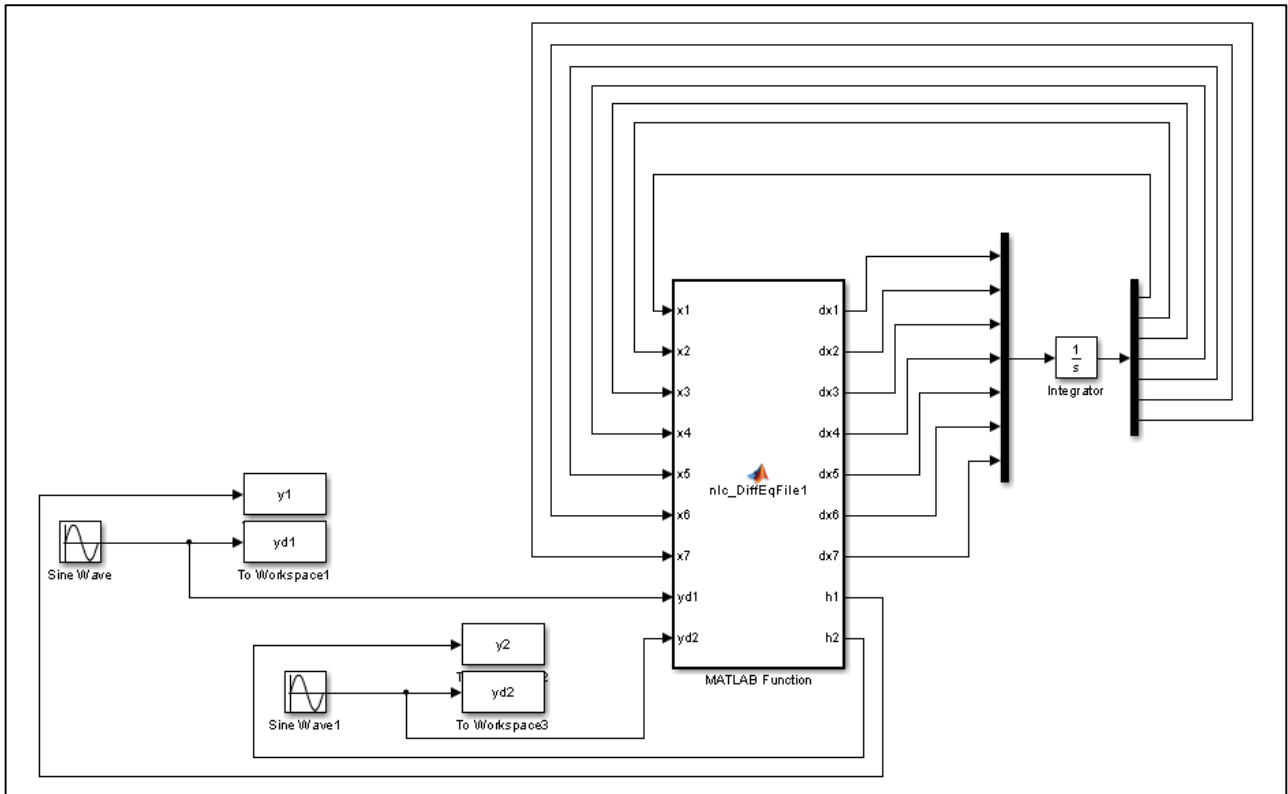


Figure 4 Simulation of Exampale 5.10.1

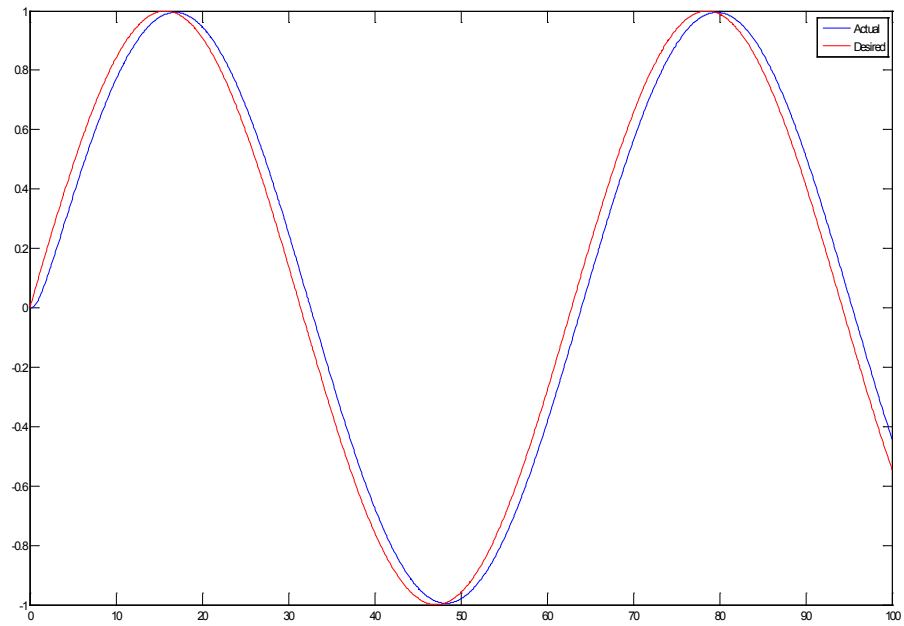


Figure 5 Simulation of the output for the system in Example 5.10.1

5.10.2 Dynamic Extension Example

Here we will show the program output when using a system shown before in as example of Dynamic Extension (see section 4.4). The dynamic equations are as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \cos x_3 & 0 \\ \sin x_3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***
```

```
Below is a list of nonlinear models where you can select to apply  
nonlinear control design. You can enter your own model as well.
```

```
For Help and more Information about this package typ 0 "zero"
```

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model
2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor
5. User Defined Model
6. General nonlinear model examples

```
Select a model by typing the corresponding number: 6
```

1. An example of Dynamic Extension Algorithm Failure (MIMO).
2. An example of Dynamic Extension Algorithm Sucess (MIMO).
3. Mechanical Arm (SISO).

4. The Contrived Model (MIMO).

5. Go back to previous menu.

Select a model by typing corresponding serial number: 2

System is affine!

States of the system are:

[x1, x2, x3]

f(x) =

0

0

0

g(x) =

[cos(x3), 0]

[sin(x3), 0]

[0, 1]

u =

u1

u2

h(x) =

x1

x2

Decoupling Matrix, E =

[cos(x3), 0]

[sin(x3), 0]

No. of States (n) is : 3

Total relative degree (r_total) is : 2

Relative degree vector (r_vector) is : [1 1]

Number of inputs (m) is : 2

Number of outputs (p) is : 2

Rank of decoupling matrix is : 1

Attention!! The system does not have a well defined relative degree.

You may consider to apply Dynamic Extension Algorithm.

NOTE: Type "nlc_DynamicExt" to start Dynamic Extension Algorithm.

Elapsed time is 4.263459 seconds.

As expected, it gives a warning message that the system is not having a well-defined relative degree because the rank of decoupling matrix is less than number of outputs. As instructed, we will type "nlc_DynamicExt" to start Dynamic Extension.

nlc_DynamicExt

Type the maximum number of iteration to run dynamic extension

NOTE: normally the maximum iteration is n (number of states). If you

choose big number, you may experience slowness in program execution

and end up with a very complicated model with many input delays!!

If you decide to choose it equal to n, leave it blank and just hit enter key!

:

.....Adding integrator to input number 1

System is affine!

New States, xn =

[x1, x2, x3, u1]

New Inputs, un =

u1

u2

New Decoupling Matrix, En =

[cos(x3), -u1*sin(x3)]

[sin(x3), u1*cos(x3)]

No. of States is : 4

Total relative degree is : 4

Relative degree vector is : [2 2]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

Full System Order Relative Degree Achieved!!

New f(x), fn =

u1*cos(x3)

u1*sin(x3)

0

0

New $g(x)$, $g_n =$

[0, 0]

[0, 0]

[0, 1]

[1, 0]

Transformation Equations, $Z =$

$z_1 = x_1$

$z_2 = u_1 \cos(x_3)$

$z_3 = x_2$

$z_4 = u_1 \sin(x_3)$

Nonlinear feedback, $U =$

$u_1 = (v_2 \sin(x_3)) / (\cos(x_3)^2 + \sin(x_3)^2) + (v_1 \cos(x_3)) / (\cos(x_3)^2 + \sin(x_3)^2)$

$u_2 = (v_2 \cos(x_3)) / (u_1 \cos(x_3)^2 + u_1 \sin(x_3)^2) - (v_1 \sin(x_3)) / (u_1 \cos(x_3)^2 + u_1 \sin(x_3)^2)$

Following is the linearized state space system:

$A =$

0 1 0 0

0 0 0 0

0 0 0 1

0 0 0 0

$B =$

0 0

1 0

0 0

0 1

C =

```
1    0    0    0
0    0    1    0
```

D =

```
0
```

Above A,B,C,D matrices can be used for linear controller desing. This package can prepare a code to be used in a simulink block "MatlabFunction" where you can perform simulation more effectively. The code will have an emeded LQI controller. To prepare the code type "nlc_WriteToFile"

Elapsed time is 1.412005 seconds.

As it can be seen from the result the program succeeded to achieve full order relative degree. Now we simulate tracking controller.

```
nlc_WriteToFile
```

```
LQI controller design....
```

```
please specify the states weight, q =: 50000
```

```
please specify the inputs gain, r =: 0.01
```

The code will be shown below, select and copy then past it in MatlabFunction block which is in the Simulink file. Once you copy press any key then the Simulink file will be opened. You can save the simulation outputs to workspace and name them as: y1,y2,.. for outputs and yd1, yd2,.. for desired ouputs. After runing the simulink you can plot the resluts using the command "nlc_PlotSimulation"

```
Press any key when your are ready...
```

```
%%%%%%%%%% Begining of the code %%%%%%%%%%
```

```
function [dx1,dx2,dx3,dx4,dx5,dx6,h1,h2] = nlc_DiffEqFile(x1,x2,x3,u1,x5,x6,yd1,yd2)
```

```
z = [ x1;u1*cos(x3);x2;u1*sin(x3);x5;x6 ];
```

```
K = [ 2302.9348,67.8666,-3.4536e-16,-1.1744e-15,-2236.068,2.4937e-12;-2.3557e-14,-  
1.1744e-15,2302.9348,67.8666,3.8044e-12,-2236.068 ];
```

```
v = -K*z;
```

```
v1 = v(1);
```

```
v2 = v(2);
```

```
mu1 = (v2*sin(x3))/(cos(x3)^2 + sin(x3)^2) + (v1*cos(x3))/(cos(x3)^2 + sin(x3)^2);
```

```
u2 = (v2*cos(x3))/(u1*cos(x3)^2 + u1*sin(x3)^2) - (v1*sin(x3))/(u1*cos(x3)^2 +  
u1*sin(x3)^2);
```

```
dx1 = u1*cos(x3);
```

```
dx2 = u1*sin(x3);
```

```
dx3 = u2;
```

```
dx4 = mu1;
```

```
h1 = x1;
```

```
h2 = x2;
```

```
dx5 = yd1 - h1;
```

```
dx6 = yd2 - h2;
```

```
%%%%%%%%%% End of the code %%%%%%%%%%
```

Copy above code and past it in MatlabFunction block. You can save

the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired ouputs. After runing the simulink

you can plot the results using the command "nlc_PlotSimulation"

Press any key when your are ready...

This may take while, be patient please....

Below figure shows the plot of the output versus desired reference.

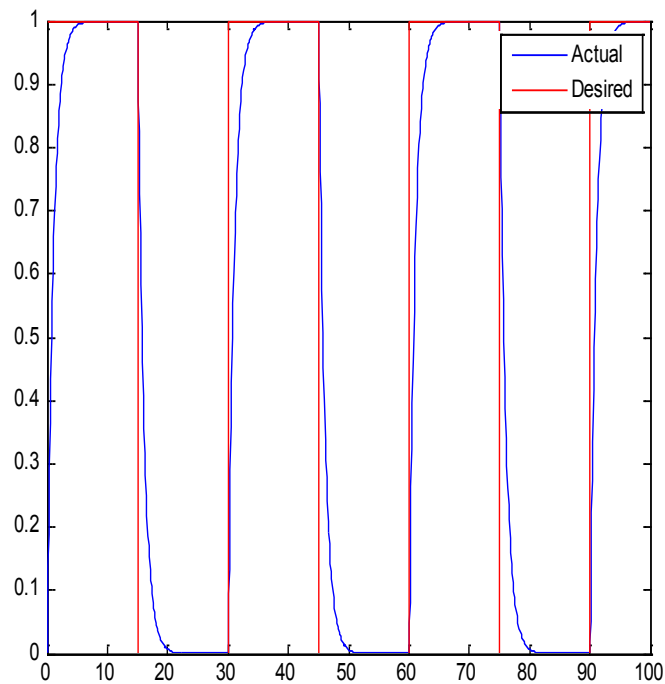


Figure 6 Simulation results for the system of example 5.10.2

5.10.3 Robot with Flexible Joint

The dynamic equations of a singular link robot arm with a revolute elastic joint rotating in a vertical plane are given by:

$$J_1 \ddot{q}_1 + F_1 \dot{q}_1 + k(q_1 - q_2) + Mgl \cdot \sin(q_1) = 0$$

$$J_m \ddot{q}_2 + F_m \dot{q}_2 - k(q_1 - q_2) = u$$

$$y = q_1$$

In which q_1 and q_2 are the link displacement and the rotor displacement, respectively. The link inertia J_1 , the motor rotor inertia J_m , the elastic constant k , the link mass M , the gravity constant g , the center of mass l and control u is the torque delivered by the motor. The objective is to design control signal u so that q_1 tracks a desired reference $q_{r1}(t)$ assuming the whole state $[q_1, \dot{q}_1, q_2, \dot{q}_2]$ is measured. The model can be written in state space format by choosing as state variables:

$$x_1 = q_1, \quad x_2 = \dot{q}_1, \quad x_3 = q_2, \quad x_4 = \dot{q}_2$$

The model is rewritten in state space form as:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -\left(\frac{F_1}{J_1}\right)x_2 - \left(\frac{Mgl}{J_1}\right)\sin(x_1) - \left(\frac{k}{J_1}\right)(x_1 - x_3)$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = -\left(\frac{F_m}{J_m}\right)x_4 + \left(\frac{k}{J_m}\right)(x_1 - x_3) + \frac{u}{J_m}$$

Now, we will run the NLC package and select user defined model to enter above state space equations.

```
*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***
```

```
Below is a list of nonlinear models where you can select to apply
```

```
nonlinear control design. You can enter your own model as well.
```

```
For Help and more Information about this package typ 0 "zero"
```

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model
2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor

5. User Defined Model

6. General nonlinear model examples

Select a model by typing the corresponding number: 5

User defined model is selected.

After 10 seconds, you will be directed to m file editor where you can type your own model follow the instruction and save the file when complete.

Come back here and press any key to continue...

It gives an instruction to wait for 10 seconds until m-file editor opens the user defined model. Here it is shown below where the robot model is entered. Notice a detailed description is give on how to enter the model.

```
function [dx,x,u,h] = nlc_UserDefinedModel
disp('          ');
%% User Defined Model:
% this function is part of NLC package (type help nlc_main for more information).
%
% (C) 2014-2015, Khalid E. Al-Khater, King Fahd University of Petrolume and
% Minerals (KFUPM), abumahdi1425@gmail.com
%
% Here you can enter you own model in a ordinary differential fromat. below
% is an example of entering the model:
%
% first of all, you need to define number of states, copy below line and
% change the assigned number to "n"
%
% n = 5;
%
% then you need to define symbols for inputs and states, you can use the
% common convention (u1, u2, u3,..x1, x2, x3,..) or use you own convention
% (F, R, mu,...,T, S,...), however, in either cases you need to save your
% states in a column vector called "x" and inputs in a column vector called
% "u". Here an example of declaring five states as (x1,x2,..,x5):
%
% x = sym(zeros(1,n));
% for countr = 1 : n
%     eval(sprintf('syms x%d', countr));
%     x(:,countr) = eval(sprintf('x%d',countr));
```

```

% end
%
% in above the variables x1,x2,...,xn already stored in coloumn vector x.
% another way to define states is shown by following example:
%
% syms T F Rd Y G1
% x = [T; F; Rd; Y; G1];
%
% you can define the inputs in the same way. for example:
%
% syms u1 u2
%
% parameters can defined as variables as well in the similar way. like the
% example below we have 15 parameters labeled as a1, a2, ..., a15 defined as
% follows:
%
% for countr = 1 : 15
%     eval(sprintf('syms a%d', countr-1));
%     a(:,countr) = eval(sprintf('a%d',countr-1));
% end
%
% The next step is to write your differential equation. The derivatives can
% are stored in vector dx, so dx1/dt is entered as dx(1) = ... and dx2/dt
% is entered as dx(2) = ...
% another way is to assign derivative to any varialbe name like dT, dF, dRd
% and then define derivatives as dx = [dT; dF; dRd; ... ]
% one way to writed differential exqaution is given below:
% dx(1) = x(2);
% dx(2) = a(1) + a(2)*x(2) + a(3)*x(2)^2 + (a(4) + a(5)*x(4)-sqrt(a(6) + ...
%     a(7)*x(4)))*x(3)^2;
% dx(3) = a(8) + a(9)*x(3) + (a(10)*sin(x(4)) + a(11))*x(3)^2+u1;
% dx(4) = x(5);
% dx(5) = a(12) + a(13)*x(4) + a(14)*x(3)^2*sin(x(4)) + a(15)*x(5) + u2;
%
% another way shown by following:
% dT = F/R + u1;
% dF = T*u2 - Y;
% dx = [dT; dF];
%
% The last step is to define ouputs and store then in store them coloumn
% vector "h", example is shown below:
%
% h = [x(1); x(3)];
%
% type your differential equations below. When you finish save this file
% and return to command window to continou the analysis. You can edit
% whatever written below or delete everything and type your model.
%
%
%% Robot with flexible joint
disp('Robot with flexible joint...');
n = 4;
syms u F1 J1 M g l k Fm Jm
x = sym(zeros(1,n));
for countr = 1 : n
    eval(sprintf('syms x%d', countr));
    x(:,countr) = eval(sprintf('x%d',countr));
end
dx(1) = x2;
dx(2) = -(F1/J1)*x2-(M*g*l/J1)*sin(x1)-(k/J1)*(x1-x3);
dx(3) = x4;

```

```

dx(4) = -(Fm/Jm)*x4+(k/Jm)*(x1-x3)+(1/Jm)*u;

% Parameters
Jm = 3.7e-3;
J1 = 9.3e-3;
M = 2.1e-1;
l = 3.1e-1;
k = 1.8e-1;
Fm = 4.6e-2;
F1 = 3.0e-2;
g = 9.8e-3;

dx = eval(dx);

h = x1;

```

after that we save the file and go back to MATLAB command window where it is waiting for user response to continue.

After 10 seconds, you will be directed to m file editor where you can type your own model

follow the instruction and save the file when complete.

Come back here and press any key to continue...

Robot with flexible joint...

System is affine!

States of the system are:

[x1, x2, x3, x4]

f(x) =

x2

(600*x3)/31 - (100*x2)/31 - (600*x1)/31 -
(1838852153772690625*sin(x1))/26805424982109192192

x4

(1800*x1)/37 - (1800*x3)/37 - (460*x4)/37

g(x) =

0

0

0

10000/37

u =

u

h(x) =

x1

Decoupling Matrix, E =

6000000/1147

No. of States (n) is : 4

Total relative degree (r_total) is : 4

Relative degree vector (r_vector) is : [4]

Number of inputs (m) is : 1

Number of outputs (p) is : 1

Rank of decoupling matrix is : 1

Transformation Equations, Z =

z1 = x1

z2 = x2

z3 = (600*x3)/31 - (100*x2)/31 - (600*x1)/31 -
(1838852153772690625*sin(x1))/26805424982109192192

$$z4 = (60000*x1)/961 + (10000*x2)/961 - (60000*x3)/961 + (600*x4)/31 + (45971303844317265625*\sin(x1))/207742043611346239488 - x2*((1838852153772690625*\cos(x1))/26805424982109192192 + 600/31)$$

Nonlinear feedback, U =

$$u1 = (1147*v1)/6000000 - (9*x1)/50 + (9*x3)/50 + (449*x4)/7750 - (1147*((1838852153772690625*\cos(x1))/26805424982109192192 + 8600/961)*((600*x1)/31 + (100*x2)/31 - (600*x3)/31 + (1838852153772690625*\sin(x1))/26805424982109192192))/6000000 - (1147*x2*((45971303844317265625*\cos(x1))/207742043611346239488 + (1838852153772690625*x2*\sin(x1))/26805424982109192192 + 60000/961))/6000000$$

Following is the linearized state space system:

A =

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

B =

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

C =

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

D =

$$0$$

Above A,B,C,D matrices can be used for linear controller desing. This package

can prepare a code to be used in a simulink block "MatlabFunction" where you

can perform simulation more effectively. The code will have an emeded LQI

controller. To prepare the code type "nlc_WriteToFile"

Elapsed time is 22.033676 seconds.

nlc_WriteToFile

LQI controller design....

please specify the states weight, q =: 50000

please specify the inputs gain, r =: 0.01

The code will be shown below, select and copy then past it in MatlabFunction block

which is in the Simulink file. Once you copy press any key then the Simulink file

will be opened. You can save the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired ouputs. After runing the simulink

you can plot the resluts using the command "nlc_PlotSimulation"

Press any key when your are ready...

%%%%%%%%% Begining of the code %%%%%%%%%

```
function [dx1,dx2,dx3,dx4,dx5,h1] = nlc_DiffEqFile(x1,x2,x3,x4,x5,yd1)
```

```
z = [ x1;x2;(600*x3)/31 - (100*x2)/31 - (600*x1)/31 -  
(1838852153772690625*sin(x1))/26805424982109192192;(60000*x1)/961 + (10000*x2)/961 -  
(60000*x3)/961 + (600*x4)/31 + (45971303844317265625*sin(x1))/207742043611346239488 -  
x2*((1838852153772690625*cos(x1))/26805424982109192192 + 600/31);x5 ];
```

```
K = [ 3084.8694,1009.9021,179.0229,18.9221,-2236.068 ];
```

```
v = -K*z;
```

```
v1 = v(1);
```

```
u = (1147*v1)/6000000 - (9*x1)/50 + (9*x3)/50 + (449*x4)/7750 -  
(1147*((1838852153772690625*cos(x1))/26805424982109192192 + 8600/961)*((600*x1)/31 +  
(100*x2)/31 - (600*x3)/31 + (1838852153772690625*sin(x1))/26805424982109192192))/6000000 -
```

```

(1147*x2*((45971303844317265625*cos(x1))/207742043611346239488 +
(1838852153772690625*x2*sin(x1))/26805424982109192192 + 60000/961))/6000000;

dx1 = x2;

dx2 = (600*x3)/31 - (100*x2)/31 - (600*x1)/31 -
(1838852153772690625*sin(x1))/26805424982109192192;

dx3 = x4;

dx4 = (10000*u)/37 + (1800*x1)/37 - (1800*x3)/37 - (460*x4)/37;

h1 = x1;

dx5 = yd1 - h1;

%%%%%%%%% End of the code %%%%%%%%%

```

Copy above code and past it in MatlabFunction block. You can save

the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired outputs. After running the simulink

you can plot the results using the command "nlc_PlotSimulation"

Press any key when your are ready...

This may take while, be patient please....

The system is having a well-defined degree equals to number of states. The linearized model used to desing LQI controller. Like previous examples the code for Simulink has been copied and below is the simulation results:

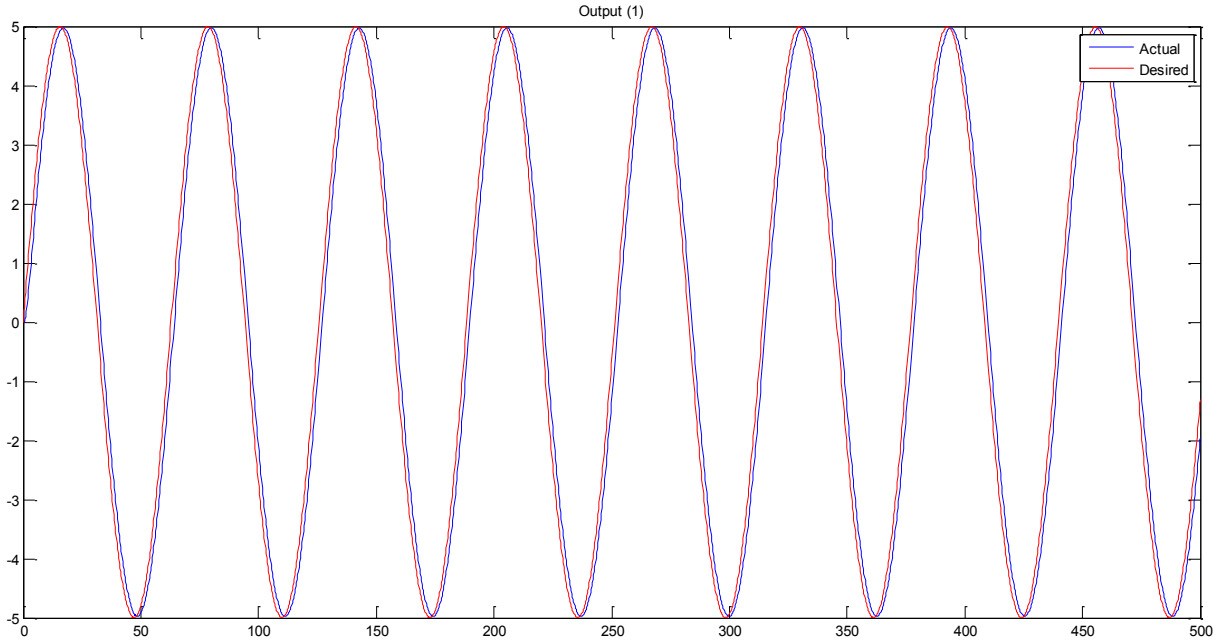


Figure 7 Simulation Results of Example 5.10.3

5.10.4 Continues Stirred Reactor Model (non-square)

Consider a CSTR in which elementary isothermal liquid phase, series reactions are being carried out. The chemical reaction system is $A + B \rightarrow C$, $A + D \rightarrow E$. The objective is to keep the concentrations of C and E at a desired set point by manipulating the inlet feed concentrations of A , B and D . The system can be written as follows:

$$\dot{x}_1 = -k_1(x_1x_2 + x_1c_{BS} + x_2c_{AS}) - k_2(x_1x_4 + x_1c_{DS} + x_4c_{AS}) - \frac{F}{V}(x_1 + u_1)$$

$$\dot{x}_2 = -k_1(x_1x_2 + x_1c_{BS} + x_2c_{AS}) - \frac{F}{V}(x_2 + u_2)$$

$$\dot{x}_3 = k_1(x_1x_2 + x_1c_{BS} + x_2c_{AS}) - \frac{F}{V}x_3$$

$$\dot{x}_4 = -k_2(x_1x_4 + x_1c_{DS} + x_4c_{AS}) - \frac{F}{V}(x_4 + u_3)$$

$$\dot{x}_5 = k_2(x_1x_4 + x_1c_{DS} + x_4c_{AS}) - \frac{F}{V}x_5$$

The program results of designing input-output linearizing control is shown below.

*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***

Below is a list of nonlinear models where you can select to apply nonlinear control design. You can enter your own model as well.

For Help and more Information about this package typ 0 "zero"

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model
2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor
5. User Defined Model
6. General nonlinear model examples

Select a model by typing the corresponding number: 3

CSTR, Series Reaction Model is selected.

System is affine!

States of the system are:

[x1, x2, x3, x4, x5]

f(x) =

$$- 4*x1 - 2*x2 - 4*x4 - x1*x2 - 2*x1*x4$$

$$- x1 - 3*x2 - x1*x2$$

$$\begin{aligned}
 & x_1 + 2x_2 - x_3 + x_1x_2 \\
 & - 2x_1 - 5x_4 - 2x_1x_4 \\
 & 2x_1 + 4x_4 - x_5 + 2x_1x_4
 \end{aligned}$$

$g(x) =$

$$[1, 0, 0]$$

$$[0, 1, 0]$$

$$[0, 0, 0]$$

$$[0, 0, 1]$$

$$[0, 0, 0]$$

$u =$

u_1

u_2

u_3

$h(x) =$

x_3

x_5

Decoupling Matrix, $E =$

$$[x_2 + 1, x_1 + 2, 0]$$

$$[2x_4 + 2, 0, 2x_1 + 4]$$

No. of States (n) is : 5

Total relative degree (r_total) is : 4

Relative degree vector (r_vector) is : [2 2]

Number of inputs (m) is : 3

Number of outputs (p) is : 2

Rank of decoupling matrix is : 2

WARNING: Total relative degree is lower than system order!!!

It is essential to analyze stability of internal dynamics!

However, Applying Dynamic Extension Algorithm may help to achieve full order relative degree.

NOTE: Type "nlc_DynamicExt" to start Dynamic Extension Algorithm.

Elapsed time is 4.970026 seconds.

As it is noticed above, the system is having total relative degree less than the number of states. Which mean the stability of internal dynamics is not guaranteed. However, we can still design the controller and since the program is designed for full order relative degree systems, we have to execute the remaining functions manually. Namely, the state transformation, control law, and state space of linearized system as it is shown below. Following is the state transformation:

```
nlc_StatTrans(f,h,x,r_vector);
```

Transformation Equations, Z =

$$z_1 = x_3$$

$$z_2 = x_1 + 2x_2 - x_3 + x_1x_2$$

$$z_3 = x_5$$

$$z_4 = 2x_1 + 4x_4 - x_5 + 2x_1x_4$$

next, is the control law and state space of linearized system:

Nonlinear feedback, $U =$

$$u1 = (((x2 + 1) * (4 * x1 + 2 * x4 + x1^2 + x4^2 + 5)) / (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24) - ((2 * x4 + 2) * (x2 + 1) * (x4 + 1)) / (2 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24))) * (v1 + x1 + 2 * x2 - x3 + (x1 + 2) * (x1 + 3 * x2 + x1 * x2) + x1 * x2 + (x2 + 1) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4)) + (((2 * x4 + 2) * (4 * x1 + 2 * x2 + x1^2 + x2^2 + 5)) / (4 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24))) - ((x2 + 1)^2 * (x4 + 1)) / (2 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24))) * (v2 + 2 * x1 + 4 * x4 - x5 + (2 * x4 + 2) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4) + 2 * x1 * x4 + (2 * x1 + 4) * (2 * x1 + 5 * x4 + 2 * x1 * x4))$$

$$u2 = ((x1 + 2) * (4 * x1 + 2 * x4 + x1^2 + x4^2 + 5) * (v1 + x1 + 2 * x2 - x3 + (x1 + 2) * (x1 + 3 * x2 + x1 * x2) + x1 * x2 + (x2 + 1) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4))) / (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24) - ((x1 + 2) * (x2 + 1) * (x4 + 1) * (v2 + 2 * x1 + 4 * x4 - x5 + (2 * x4 + 2) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4) + 2 * x1 * x4 + (2 * x1 + 4) * (2 * x1 + 5 * x4 + 2 * x1 * x4))) / (2 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24))$$

$$u3 = ((2 * x1 + 4) * (4 * x1 + 2 * x2 + x1^2 + x2^2 + 5) * (v2 + 2 * x1 + 4 * x4 - x5 + (2 * x4 + 2) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4) + 2 * x1 * x4 + (2 * x1 + 4) * (2 * x1 + 5 * x4 + 2 * x1 * x4))) / (4 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24)) - ((2 * x1 + 4) * (x2 + 1) * (x4 + 1) * (v1 + x1 + 2 * x2 - x3 + (x1 + 2) * (x1 + 3 * x2 + x1 * x2) + x1 * x2 + (x2 + 1) * (4 * x1 + 2 * x2 + 4 * x4 + x1 * x2 + 2 * x1 * x4))) / (2 * (40 * x1 + 8 * x2 + 8 * x4 + x1^2 * x2^2 + x1^2 * x4^2 + 8 * x1 * x2 + 8 * x1 * x4 + 4 * x1 * x2^2 + 2 * x1^2 * x2 + 4 * x1 * x4^2 + 2 * x1^2 * x4 + 26 * x1^2 + 8 * x1^3 + 4 * x2^2 + x1^4 + 4 * x4^2 + 24))$$

`nlc_EquLinSys(r_vector);`

$A =$

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

$B =$

$$\begin{bmatrix} 0 & 0 \end{bmatrix}$$

```

    1    0
    0    0
    0    1
C =
    1    0    0    0
    0    0    1    0
D =
    0

```

Now we can design LQI for linearized system:

```
nlc_WriteToFile
```

```
LQI controller design....
```

```
please specify the states weight, q =: 50000
```

```
please specify the inputs gain, r =: 0.01
```

The code will be shown below, select and copy then past it in MatlabFunction block which is in the Simulink file. Once you copy press any key then the Simulink file will be opened. You can save the simulation outputs to workspace and name them as: y1,y2,.. for outputs and yd1, yd2,... for desired outputs. After running the simulink you can plot the results using the command "nlc_PlotSimulation"

```
Press any key when your are ready...
```

```
%%%%%%%%% Begining of the code %%%%%%%%%
```

```
function [dx1,dx2,dx3,dx4,dx5,dx6,dx7,h1,h2] =
nlc_DiffEqFile(x1,x2,x3,x4,x5,x6,x7,yd1,yd2)
```

```
z = [ x3;x1 + 2*x2 - x3 + x1*x2;x5;2*x1 + 4*x4 - x5 + 2*x1*x4;x6;x7 ];
```

K = [2302.9348, 67.8666, -3.4536e-16, -1.1744e-15, -2236.068, 2.4937e-12; -2.3557e-14, -1.1744e-15, 2302.9348, 67.8666, 3.8044e-12, -2236.068];

v = -K*z;

v1 = v(1);

v2 = v(2);

u1 = (((x2 + 1)*(4*x1 + 2*x4 + x1^2 + x4^2 + 5))/(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24) - ((2*x4 + 2)*(x2 + 1)*(x4 + 1))/(2*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24)))*(v1 + x1 + 2*x2 - x3 + (x1 + 2)*(x1 + 3*x2 + x1*x2) + x1*x2 + (x2 + 1)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4)) + (((2*x4 + 2)*(4*x1 + 2*x2 + x1^2 + x2^2 + 5))/(4*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24)) - ((x2 + 1)^2*(x4 + 1))/(2*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24)))*(v2 + 2*x1 + 4*x4 - x5 + (2*x4 + 2)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4) + 2*x1*x4 + (2*x1 + 4)*(2*x1 + 5*x4 + 2*x1*x4));

u2 = ((x1 + 2)*(4*x1 + 2*x4 + x1^2 + x4^2 + 5)*(v1 + x1 + 2*x2 - x3 + (x1 + 2)*(x1 + 3*x2 + x1*x2) + x1*x2 + (x2 + 1)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4)))/(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24) - ((x1 + 2)*(x2 + 1)*(x4 + 1)*(v2 + 2*x1 + 4*x4 - x5 + (2*x4 + 2)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4) + 2*x1*x4 + (2*x1 + 4)*(2*x1 + 5*x4 + 2*x1*x4)))/(2*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24));

u3 = ((2*x1 + 4)*(4*x1 + 2*x2 + x1^2 + x2^2 + 5)*(v2 + 2*x1 + 4*x4 - x5 + (2*x4 + 2)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4) + 2*x1*x4 + (2*x1 + 4)*(2*x1 + 5*x4 + 2*x1*x4)))/(4*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24)) - ((2*x1 + 4)*(x2 + 1)*(x4 + 1)*(v1 + x1 + 2*x2 - x3 + (x1 + 2)*(x1 + 3*x2 + x1*x2) + x1*x2 + (x2 + 1)*(4*x1 + 2*x2 + 4*x4 + x1*x2 + 2*x1*x4)))/(2*(40*x1 + 8*x2 + 8*x4 + x1^2*x2^2 + x1^2*x4^2 + 8*x1*x2 + 8*x1*x4 + 4*x1*x2^2 + 2*x1^2*x2 + 4*x1*x4^2 + 2*x1^2*x4 + 26*x1^2 + 8*x1^3 + 4*x2^2 + x1^4 + 4*x4^2 + 24));

dx1 = u1 - 4*x1 - 2*x2 - 4*x4 - x1*x2 - 2*x1*x4;

dx2 = u2 - x1 - 3*x2 - x1*x2;

dx3 = x1 + 2*x2 - x3 + x1*x2;

dx4 = u3 - 2*x1 - 5*x4 - 2*x1*x4;

dx5 = 2*x1 + 4*x4 - x5 + 2*x1*x4;

h1 = x3;

```
h2 = x5;  
  
dx6 = yd1 - h1;  
  
dx7 = yd2 - h2;  
  
%%%%%%%%% End of the code %%%%%%%%%
```

Copy above code and past it in MatlabFunction block. You can save

the simulation outputs to workspace and name them as:

y1,y2,.. for outputs and yd1, yd2,... for desired outputs. After running the simulink

you can plot the results using the command "nlc_PlotSimulation"

Press any key when your are ready...

This may take while, be patient please....

Below is the simulation results:

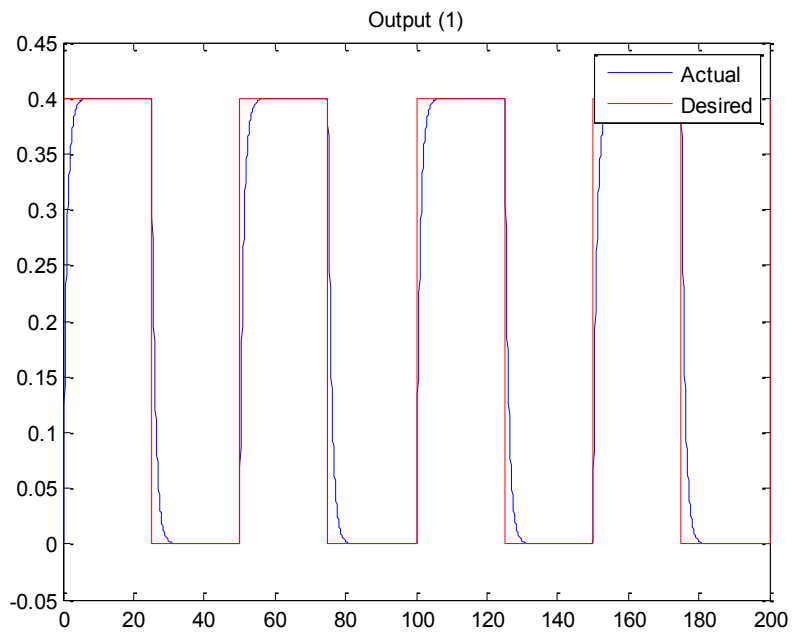


Figure 8 Simulation Results of Example 5.10.4 (Output 1)

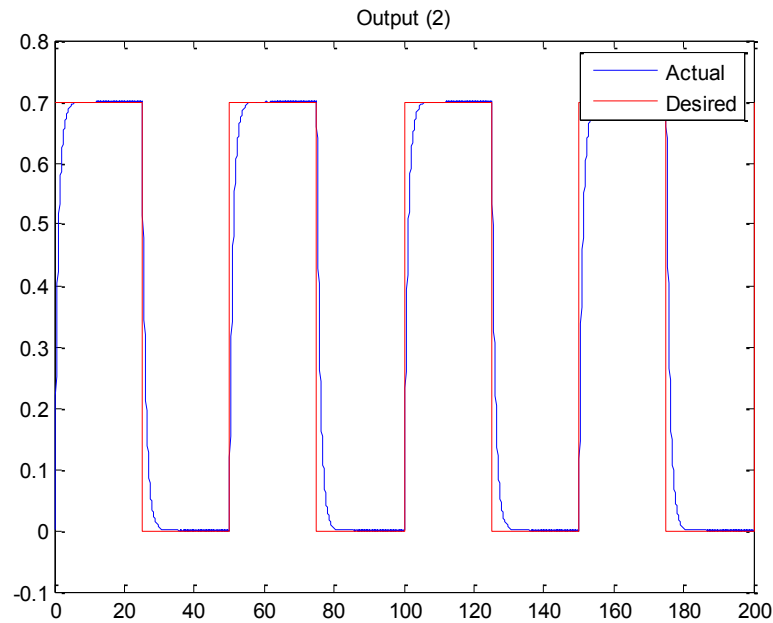


Figure 9 Simulation Results of Example 5.10.4 (Output 2)

5.10.5 Polymerization Fluidized Bed Reactor Model

Now we are going to have another example where dynamic extension succeeded but with more complicated model from chemical process industry. A fluidized bed reactor (FBR) is a type of reactor device that can be used to carry out a variety of multiphase chemical reactions. In this type of reactor, a fluid (gas or liquid) is passed through a rough solid material (usually a catalyst possibly shaped as tiny spheres) at high enough velocities to suspend the solid and cause it to behave as though it were a fluid. This process, known as fluidization, conveys many important advantages to the FBR. As a result, the fluidized bed reactor is now used in many industrial applications. Fluidization occurs when small solid particles are suspended in an upward flowing stream of fluid. The fluid velocity is sufficient to suspend the particles, but it is not large enough to carry them out of the vessel. The solid particles swirl around the bed rapidly, creating excellent mixing among them. The material “fluidized” is usually a solid and the “fluidizing medium” is either a liquid or gas. The characteristics and behavior of a fluidized bed are strongly dependent on both the solid and liquid or gas properties. Many models developed for FBR, e.g. two-phase model and well-mixed model. We

will present here two well-mixed models one with first order model approximation for heat exchanger and the other one with staged heat exchanger model. The nonlinear model presented by set of differential equations and another set of algebraic equations describing the dynamic behavior of the reactor as well as heat and mass balance. Following are the list of those equations. Description of symbols and algebraic equation are listed in table 1.

- Monomer Mass balance:

$$\frac{dM_1}{dt} = \frac{1}{Vg} (F_{M1f} - b_{M1} - R_{M1})$$

- Catalyst Mass Balance:

$$\frac{dy}{dt} = F_y - k_d y - y \left(\frac{O_p}{B_w} \right)$$

- Energy Balance (to be used when first order heat exchanger model is considered):

$$\frac{dT}{dt} = \frac{H_f - Q_d + H_r - H_p + H_c}{M_r C_{pr} + B_w C_{ppol}}$$

- Inert Gas Mass Balance:

$$\frac{dI}{dt} = \frac{1}{V_g} (F_{If} - b_I)$$

- First Order Heat Exchanger Model

$$\frac{dQ_d}{dt} = \frac{F_g C_{pg} (T - T_{g0ss}) - Q_d}{\tau}$$

- Staged Heat Exchanger Model

$$\frac{dT_{w1}}{dt} = \frac{F_w}{M_w} (T_{wi} - T_{w1}) - \frac{AU}{M_w C_{pw}} (T_w - T_{g1})$$

$$\frac{dT_{g1}}{dt} = \frac{F_g}{M_g} (T - T_{g1}) - \frac{AU}{M_g C_{pg}} (T_{w1} - T_{g1})$$

- Energy Balance (to be used when staged heat exchanger model is considered)

$$\frac{dT}{dt} = \frac{H_f + H_{g0} - H_{top} + H_r - H_p + H_c}{M_r C_{pr} + B_w C_{ppol}}$$

Table 1 List of all symbols and algebraic equations for FBR model

Symbol	Definition and related algebraic equation
M_1	Monomer concentration, $\frac{mol}{m^3}$
V_g	Volume of gas phase in the reactor, m^3
F_{M1f}	Molar feed rate of monomer to reactor, $\frac{mol}{s}$
b_{M1}	Outflow rate of monomer in the bleed stream, $b_{M1} = \frac{M_1}{M_1 + I} b_t$
R_{M1}	Rate of monomer consumption due to reaction
b_t	Total bleed stream flow rate that depends on the reaction pressure, $b_t = v_p C_v \sqrt{P - P_v}$
I	Concentration of inert gas, $\frac{mol}{m^3}$
v_p	Bleed stream valve position
C_v	Valve coefficient
P_v	Pressure downstream of bleed valve
P	Reactor Pressure, $P = (M_1 + I)RT$
R	Ideal gas constant
T	Reactor Temperatur
R_{M1}	Monomer Ration, $R_{M1} = M_1 k_p(T) y$
k_p	Temperature independent propagation rate constant, $k_p(T) = k_p T_{ref} e^{\left(-\frac{E_a}{R}\right)\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)}$
E_a	Activation energy for propagation
T_{ref}	Reference Temperature
F_y	Molar feed rate of catalyst, $F_y = F_c a_c$
F_c	Mass flow rate of catalyst to reactor, $\frac{kg}{s}$

Symbol	Definition and related algebraic equation
a_c	Active site concentration on the catalyst, $\frac{mol}{kg}$
O_p	Outflow rate of the polymer product from the react, $O_p = R_{M1}m_{w1}$
k_d	Deactivation rate constant, $\frac{1}{s}$
m_{w1}	Molecular weight of monomer, $\frac{kg}{mol}$
B_w	Mass of polymer in the reactor, kg
H_f	Enthalpy of gas entering in the fresh feed stream, $\frac{J}{kg}$, $H_f = F_{M1f}C_{pf}(T_f - T_{ref})$
C_{pf}	Specific heat capacity of monomer, $\frac{J}{mol.K}$
T_f	Feed Temperature, K
Q_d	Heat removal rate (refer to heat exchanger model)
H_r	The rate of heat generation by reaction
ΔH_r	Enthalpy of reaction, $\frac{J}{kg}$
H_p	Enthalpy associated with polymer leaving reactor, $H_p = O_p C_{ppol}(T - T_{ref})$
C_{ppol}	Specific heat capacity of polymer, $\frac{J}{kg.K}$
H_c	Enthalpy associated with catalyst enters the reactor, $H_c = F_c C_{pc}(T_{fc} - T_{ref})$
C_{pc}	Specific heat capacity of catalyst, $\frac{J}{kg.K}$
T_{fc}	Catalyst feed temperature
$M_r C_{pr}$	Thermal capacitance of polymer, $\frac{J}{K}$
F_{If}	Molar flow rate of inert gas, $\frac{mol}{s}$
b_I	Mole fraction of inert gas, $b_I = \frac{I}{M_1+I} b_t$
F_g	Recycle gas flow rate, $\frac{mol}{s}$
C_{pg}	Specific heat capacity of gas
τ	The first order time constant for the exchanger

Symbol	Definition and related algebraic equation
T_{g0ss}	The gas side outlet temperature, $T_{g0ss} = \frac{T_w(1-e^\gamma) - T\left(1 - \frac{F_g C_{pg}}{F_w C_{pw}}\right)}{\frac{F_g C_{pg}}{F_w C_{pw}} - e^\gamma}$
T_w	Cooling water temperature, K
F_w	Cooling water flow, kg/s
C_{pw}	Cooling water heat capacity, $\frac{J}{s.K}$
γ	$\gamma = AU \left(\frac{1}{F_g C_{pg}} + \frac{1}{F_w C_{pw}} \right)$
P_{M1}	Monomer partial pressure, $P_{M1} = RM_1 T$
P_r	Production rate, $P_r = k_p y M_1 m_{w1}$
M_w	Mass holdup of cooling water in heat exchanger, kg
T_{wi}	Inlet cooling water temperature to heat exchanger, K
M_g	Mass holdup of recycle gas in heat exchanger, kg
C_{pg}	Specific heat capacity of recycle gas, $\frac{J}{kg.K}$
H_{top}	The enthalpy of the gas leaving the reactor, $H_{top} = (F_g + b_t) C_{pg} (T - T_{ref})$
H_{g0}	the enthalpy of the recycle stream entering the reactor, $H_{g0} = H_{top} - H_b - Q_d$

Below is the program results when FBR with fist order heat exchanger model is selected.

*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***

Below is a list of nonlinear models where you can select to apply

nonlinear control design. You can enter your own model as well.

For Help and more Information about this package typ 0 "zero"

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model

2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continuous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor
5. Semibatch Co-polymerization Reactor
6. User Defined Model
7. General nonlinear model examples

Select a model by typing the corresponding number: 1

FBR with 1st order heat exchanger model is chosen.

System is not affine!

...Changing to affine by adding integrator(s) to the non-affine input(s).

System is affine!

States of the system are:

[M1, Y, T, I1, Qd, Fw]

f(x) =

$$-(M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R) - FM1f + (Cv*M1*Vp*(RR*T*(I1 + M1) - Pv)^(1/2))/(I1 + M1))/Vg$$

$$- Y*kd - (M1*Y^2*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R))/Bw$$

$$-(Qd - CpIn*FIf*(Tf - Tref) - Cpf*FM1f*(Tf - Tref) + DeltaHR*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref))/(MrCpr + Bw*Cpol)$$

$$(FIf - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^(1/2))/(I1 + M1))/Vg$$

$$-(Qd - Fg*(T - (Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw))*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/tau$$

0

$g(x) =$

[0, 0]

[ac, 0]

[0, 0]

[0, 0]

[0, 0]

[0, 1]

$u =$

Fc

alpha1

$h(x) =$

M1

T

Decoupling Matrix, E =

$[-(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg, 0]$

$[-(ac*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(MrCpr + Bw*Cppol), 0]$

No. of States (n) is : 6

Total relative degree (r_total) is : 4

Relative degree vector (r_vector) is : [2 2]

Number of inputs (m) is : 2

Number of outputs (p) is : 2

Rank of decoupling matrix is : 1

Attention!! The system does not have a well defined relative degree.

You may consider to apply Dynamic Extension Algorithm.

NOTE: Type "nlc_DynamicExt" to start Dynamic Extension Algorithm.

Elapsed time is 8.279945 seconds.

The system doesn't have a well-defined relative degree, hence the dynamic extension will be executed.

nlc_DynamicExt

Type the maximum number of iteration to run dynamic extension

NOTE: normally the maximum iteration is n (number of states). If you

choose big number, you may experience slowness in program execution

and end up with a very complicated model with many input delays!!

If you decide to choose it equal to n, leave it blank and just hit enter key!

:

.....Adding integrator to input number 1

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc]

New Inputs, un =

mul

alpha1

New Decoupling Matrix, En =

$$\begin{bmatrix} (M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg, \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} -(ac*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(MrCpr + Bw*Cppol), & (Fg*((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - ((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cppol)) \end{bmatrix}$$

No. of States is : 7

Total relative degree is : 6

Relative degree vector is : [3 3]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

.....Adding integrator to input number 1

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc, mul]

New Inputs, un =

mu2

alpha1

New Decoupling Matrix, En =

$$\begin{aligned} & [-(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg, -(Fg*((Cv*M1*RR*Vp)/(2*(RR*T*(I1 + M1) - \\ & Pv)^{1/2})) + (Ea*M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R))/(R*T^2))*(((Cpw*Fg*T*((CpIn*I1)/(I1 \\ & + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - \\ & (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + \\ & 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + \\ & 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) \\ & + (Cpf*M1)/(I1 + M1)))/Fw - 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + \\ & M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(Vg*tau*(MrCpr + Bw*Cppol))] \end{aligned}$$

$$\begin{aligned} & [\\ & 0, \\ & (Fg*((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - \\ & (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + \\ & M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - \\ & (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - \\ & 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - \\ & (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + \\ & (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cppol))] \end{aligned}$$

No. of States is : 8

Total relative degree is : 7

Relative degree vector is : [4 3]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

.....Adding integrator to input number 2

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc, mu1, alpha1]

New Inputs, un =

mu2

mu3

New Decoupling Matrix, En =

[
(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg,
0]

[-(ac*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(MrCpr + Bw*Cppol), (Fg*((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - ((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cppol))]

No. of States is : 9

Total relative degree is : 8

Relative degree vector is : [4 4]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

.....Adding integrator to input number 1

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc, mu1, alpha1, mu2]

New Inputs, un =

mu4

mu3

New Decoupling Matrix, En =

[-(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg, -(Fg*((Cv*M1*RR*Vp)/(2*(RR*T*(I1 + M1) - Pv)^(1/2)) + (Ea*M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R))/(R*T^2))*(((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1)))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(Vg*tau*(MrCpr + Bw*Cppl))]

[
0,
(Fg*((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1)))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cppl))]

No. of States is : 10

Total relative degree is : 9

Relative degree vector is : [5 4]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

.....Adding integrator to input number 2

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc, mu1, alpha1, mu2, mu3]

New Inputs, un =

mu4

mu5

New Decoupling Matrix, En =

[
(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg,
0]

[-(ac*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(MrCpr + Bw*Cpol), (Fg*((Cpw*Fg*T*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - ((Cpw*Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1))/(exp(AU*(Cpw/Fw + 1/(Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2*((Cpin*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cpol))]

No. of States is : 11

Total relative degree is : 10

Relative degree vector is : [5 5]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

.....Adding integrator to input number 1

System is affine!

New States, xn =

[M1, Y, T, I1, Qd, Fw, Fc, mu1, alpha1, mu2, mu3, mu4]

New Inputs, un =

mu6

mu5

New Decoupling Matrix, En =

[-(M1*ac*kp*exp(-(Ea*(1/T - 1/Tref))/R))/Vg, -(Fg*((Cv*M1*RR*Vp)/(2*(RR*T*(I1 + M1) - Pv)^(1/2)) + (Ea*M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R))/(R*T^2)))*(((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1)))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(Vg*tau*(MrCpr + Bw*Cppl)]

[0, (Fg*((Cpw*Fg*T*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*Tw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw) - (((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw^2 - (AU*Cpw*exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))/Fw^2)*(Tw*(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - 1) - T*((Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw - 1)))/(exp(AU*(Cpw/Fw + 1/(Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))))) - (Cpw*Fg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/Fw)^2)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))/(tau*(MrCpr + Bw*Cppl)]

No. of States is : 12

Total relative degree is : 11

Relative degree vector is : [6 5]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

Reached to maximum number of iterations!! Relative Degree cannot be achieved.

Elapsed time is 120.575130 seconds.

As we can see in the above results, the program failed to achieve full order relative degree. Now we will run the program selecting FBR with staged exchange heater model.

*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM ***

Below is a list of nonlinear models where you can select to apply nonlinear control design. You can enter your own model as well.

For Help and more Information about this package typ 0 "zero"

1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model
2. Fluidized Bed Reactor (FBR) with staged heat exchanger model
3. Continous Stirred Tank Reactor (CSTR), Series Reaction (non-square).
4. Induction Motor
5. Semibatch Co-polymerization Reactor
6. User Defined Model
7. General nonlinear model examples

Select a model by typing the corresponding number: 2

FBR with staged heat exchanger model is chosen.

System is affine!

States of the system are:

[M1, I1, Y, T, Tw1, Tg1]

f(x) =

$$(FIf - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1))/Vg$$

$$-(M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R) - FM1f + (Cv*M1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1))/Vg$$

$$- Y*kd - (M1*Y^2*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R))/Bw$$

$$(CpIn*FIf*(Tf - Tref) + Cpf*FM1f*(Tf - Tref) - (Fg + Cv*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})*(T - Tf)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)) + Fg*(Tg1 - Tref)*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)) - DeltaHR*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) - Cppol*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref))/(MrCpr + Bw*Cppol)$$

$$(AU*(Tg1 - Tw1))/(Cpw*Mw)$$

$$(Fg*(T - Tg1))/Mg - (AU*(Tg1 - Tw1))/(Mg*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)))$$

g(x) =

$$[0, 0]$$

$$[0, 0]$$

$$[ac, 0]$$

$$[0, 0]$$

$$[0, -(Tw1 - Twi)/Mw]$$

$$[0, 0]$$

u =

Fc

Fw

h(x) =

T

M1

Decoupling Matrix, E =

```
[  
- (ac * (DeltaHR * M1 * kp * mw1 * exp(- (Ea * (1/T - 1/Tref)) / R) + Cppol * M1 * kp * mw1 * exp(- (Ea * (1/T -  
1/Tref)) / R) * (T - Tref))) / (MrCpr + Bw * Cppol), 0]
```

```
[ ac * ((M1 * kp * exp(- (Ea * (1/T - 1/Tref)) / R) * ((Cv * Vp * (RR * T * (I1 + M1) - Pv) ^ (1/2)) / (I1 + M1) -  
(Cv * I1 * Vp * (RR * T * (I1 + M1) - Pv) ^ (1/2)) / (I1 + M1) ^ 2 + (Cv * I1 * RR * T * Vp) / (2 * (RR * T * (I1 + M1) -  
Pv) ^ (1/2) * (I1 + M1)))) / Vg ^ 2 + (Cv * I1 * RR * Vp * (DeltaHR * M1 * kp * mw1 * exp(- (Ea * (1/T - 1/Tref)) / R)  
+ Cppol * M1 * kp * mw1 * exp(- (Ea * (1/T - 1/Tref)) / R) * (T - Tref))) / (2 * Vg * (MrCpr +  
Bw * Cppol) * (RR * T * (I1 + M1) - Pv) ^ (1/2))), 0]
```

No. of States (n) is : 6

Total relative degree (r_total) is : 5

Relative degree vector (r_vector) is : [2 3]

Number of inputs (m) is : 2

Number of outputs (p) is : 2

Rank of decoupling matrix is : 1

Attention!! The system does not have a well defined relative degree.

You may consider to apply Dynamic Extension Algorithm.

NOTE: Type "nlc_DynamicExt" to start Dynamic Extension Algorithm.

Elapsed time is 6.489193 seconds.

nlc_DynamicExt

Type the maximum number of iteration to run dynamic extension

NOTE: normally the maximum iteration is n (number of states). If you

choose big number, you may experience slowness in program execution

and end up with a very complicated model with many input delays!!

If you decide to choose it equal to n, leave it blank and just hit enter key!

:

.....Adding integrator to input number 1

System is affine!

New States, xn =

[M1, I1, Y, T, Tw1, Tg1, Fc]

New Inputs, un =

mul

Fw

New Decoupling Matrix, En =

[
-(ac*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(MrCpr + Bw*Cppol),
(AU*Fg*(Tw1 - Twi))/(Mg*Mw*(MrCpr + Bw*Cppol))]

[ac*((M1*kp*exp(-(Ea*(1/T - 1/Tref))/R))*((Cv*Vp*(RR*T*(I1 + M1) - Pv)^(1/2))/(I1 + M1) - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^(1/2))/(I1 + M1)^2 + (Cv*I1*RR*T*Vp)/(2*(RR*T*(I1 + M1) - Pv)^(1/2)*(I1 + M1))))/Vg^2 + (Cv*I1*RR*Vp*(DeltaHR*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) + Cppol*M1*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(2*Vg*(MrCpr + Bw*Cppol)*(RR*T*(I1 + M1) - Pv)^(1/2)), (AU*Cv*Fg*I1*RR*Vp*(Tw1 - Twi))/(2*Mg*Mw*Vg*(MrCpr + Bw*Cppol)*(RR*T*(I1 + M1) - Pv)^(1/2))]

No. of States is : 7

Total relative degree is : 7

Relative degree vector is : [3 4]

Number of inputs and outputs is : 2

Rank of decoupling matrix is : 2

Full System Order Relative Degree Achieved!!

New f(x), fn =

$$(F_{if} - (C_v \cdot I_1 \cdot V_p \cdot (RR \cdot T \cdot (I_1 + M_1) - P_v)^{1/2}) / (I_1 + M_1)) / V_g$$

$$-(M_1 \cdot Y \cdot k_p \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R) - F_{M1f} + (C_v \cdot M_1 \cdot V_p \cdot (RR \cdot T \cdot (I_1 + M_1) - P_v)^{1/2}) / (I_1 + M_1)) / V_g$$

$$F_c \cdot a_c - Y \cdot k_d - (M_1 \cdot Y^2 \cdot k_p \cdot m_w \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R)) / B_w$$

$$(C_{pIn} \cdot F_{if} \cdot (T_f - T_{ref}) + C_{pf} \cdot F_{M1f} \cdot (T_f - T_{ref}) - (F_g + C_v \cdot V_p \cdot (RR \cdot T \cdot (I_1 + M_1) - P_v)^{1/2})) \cdot (T - T_f) \cdot ((C_{pIn} \cdot I_1) / (I_1 + M_1) + (C_{pf} \cdot M_1) / (I_1 + M_1)) + F_g \cdot (T_g1 - T_{ref}) \cdot ((C_{pIn} \cdot I_1) / (I_1 + M_1) + (C_{pf} \cdot M_1) / (I_1 + M_1)) - \Delta HR \cdot M_1 \cdot Y \cdot k_p \cdot m_w \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R) - C_{ppol} \cdot M_1 \cdot Y \cdot k_p \cdot m_w \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R) \cdot (T - T_{ref})) / (M_r C_{pr} + B_w \cdot C_{ppol})$$

$$(AU \cdot (T_{g1} - T_{w1})) / (C_{pw} \cdot M_w)$$

$$(F_g \cdot (T - T_{g1})) / M_g - (AU \cdot (T_{g1} - T_{w1})) / (M_g \cdot ((C_{pIn} \cdot I_1) / (I_1 + M_1) + (C_{pf} \cdot M_1) / (I_1 + M_1)))$$

0

New g(x), gn =

$$[0, \quad 0]$$

$$[0, \quad 0]$$

$$[0, \quad 0]$$

$$[0, \quad 0]$$

$$[0, \quad -(T_{w1} - T_{wi}) / M_w]$$

$$[0, \quad 0]$$

$$[1, \quad 0]$$

Transformation Equations, Z =

$$z_1 = T$$

$$z_2 = (C_{pIn} \cdot F_{if} \cdot (T_f - T_{ref}) + C_{pf} \cdot F_{M1f} \cdot (T_f - T_{ref}) - (F_g + C_v \cdot V_p \cdot (RR \cdot T \cdot (I_1 + M_1) - P_v)^{1/2})) \cdot (T - T_f) \cdot ((C_{pIn} \cdot I_1) / (I_1 + M_1) + (C_{pf} \cdot M_1) / (I_1 + M_1)) + F_g \cdot (T_{g1} - T_{ref}) \cdot ((C_{pIn} \cdot I_1) / (I_1 + M_1) + (C_{pf} \cdot M_1) / (I_1 + M_1)) - \Delta HR \cdot M_1 \cdot Y \cdot k_p \cdot m_w \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R) - C_{ppol} \cdot M_1 \cdot Y \cdot k_p \cdot m_w \cdot \exp(-(E_a \cdot (1/T - 1/T_{ref})) / R) \cdot (T - T_{ref})) / (M_r C_{pr} + B_w \cdot C_{ppol})$$

z3 = ...WARNING!! The formula is too long and will not be displayed

$$z4 = M1$$

$$z5 = (Fif - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1))/Vg$$

$$z6 = ((M1*Y*kp*exp(-(Ea*(1/T - 1/Tref))/R) - FM1f + (Cv*M1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1))*((Cv*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1) - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1)^2 + (Cv*I1*RR*T*Vp)/(2*(RR*T*(I1 + M1) - Pv)^{(1/2)}*(I1 + M1))))/Vg^2 + (((Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1)^2 - (Cv*I1*RR*T*Vp)/(2*(RR*T*(I1 + M1) - Pv)^{(1/2)}*(I1 + M1)))*(Fif - (Cv*I1*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)})/(I1 + M1)))/Vg^2 - (Cv*I1*RR*T*Vp*(CpIn*Fif*(Tf - Tref) + Cpf*FM1f*(Tf - Tref) - (Fg + Cv*Vp*(RR*T*(I1 + M1) - Pv)^{(1/2)}*(T - Tf))*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)) + Fg*(Tg1 - Tref))*((CpIn*I1)/(I1 + M1) + (Cpf*M1)/(I1 + M1)) - DeltaHR*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R) - Cppol*M1*Y*kp*mw1*exp(-(Ea*(1/T - 1/Tref))/R)*(T - Tref)))/(2*Vg*(MrCpr + Bw*Cpol)*(RR*T*(I1 + M1) - Pv)^{(1/2)})$$

$$z7 = \dots\text{WARNING!! The formula is too long and will not be displayed}$$

Nonlinear feedback, U =

$$u1 = \dots\text{WARNING!! The formula is too long and will not be displayed}$$

$$u2 = \dots\text{WARNING!! The formula is too long and will not be displayed}$$

Following is the linearized state space system:

A =

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

B =

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
1    0
0    0
0    0
0    0
0    1
```

C =

```
1    0    0    0    0    0    0
0    0    0    1    0    0    0
```

D =

```
0
```

Above A,B,C,D matrices can be used for linear controller desing. This package can prepare a code to be used in a simulink block "MatlabFunction" where you can perform simulation more effectively. The code will have an emeded LQI controller. To prepare the code type "nlc_WriteToFile"

Elapsed time is 5.822795 seconds.

After executing dynamic extension, the program succeeded to achieve full relative degree and computed the control law. But due to complexity of the system, the control law formula is too long and not recommended to be used since it will cause a lot of delay.

CHAPTER 6: Conclusions and Recommendations

6.1 Conclusions

The collection of functions/ subroutines developed in this thesis as NLC package forms a valuable contribution to the improvement of systematic design methodologies for nonlinear control systems. The NLC package offers powerful computational tools which provide options for symbolic analysis and design of nonlinear control systems. It provides a reliable treatment of important field of interest in the (analytic) analysis and design of nonlinear control systems: the exact linearization theories. It is made possible to compute and analyze symbolically for a class of (MIMO as well as SISO) systems for which the relative degree can be defined, the state transformation, an exact linearization of the input-output equations, and automatic code generation for simulation including linear controller design. The results established so far with this package seem to be promising, yet its value should be verified in practice, when elaborate tests with a real implementations in industry plants, robots, electrical systems, etc.

An important shortcoming in this MATLAB package is dealing with large scale problems like distillation columns where number of states is huge, and difficulties dealing with a very lengthy symbolic expressions.

6.2 Recommendations and Future Work

Fore future research it is recommended to provide extensions of the existing NLC package by the following:

- Implementation of additional control objectives in order to shift the attention from specific analysis to more complete synthesis of control problems. One may consider to add sliding mode control or adaptive control as an option for different control objectives.
- Improvement of the possibilities for symbolic evaluation of the MATLAB results.
- Improvement of the subroutines to become more user-friendly by offering choices whether or not some tests or parts of the program should be performed, or by providing better help and guidance during program execution.
- Optimization of messages, in order to return as much information as possible.
- Performing elaborate tests on existing NLC package with a number of different system models, in which realistic models should be incorporated.
- Assessment of the control approaches introduced in this thesis and implemented in NLC package when they are imposed on real control problems.
- Considering zero-dynamic analysis and to compute state transformation for systems having total relative degree less than the number of states.
- Providing more analysis and tests of non-square systems.

References

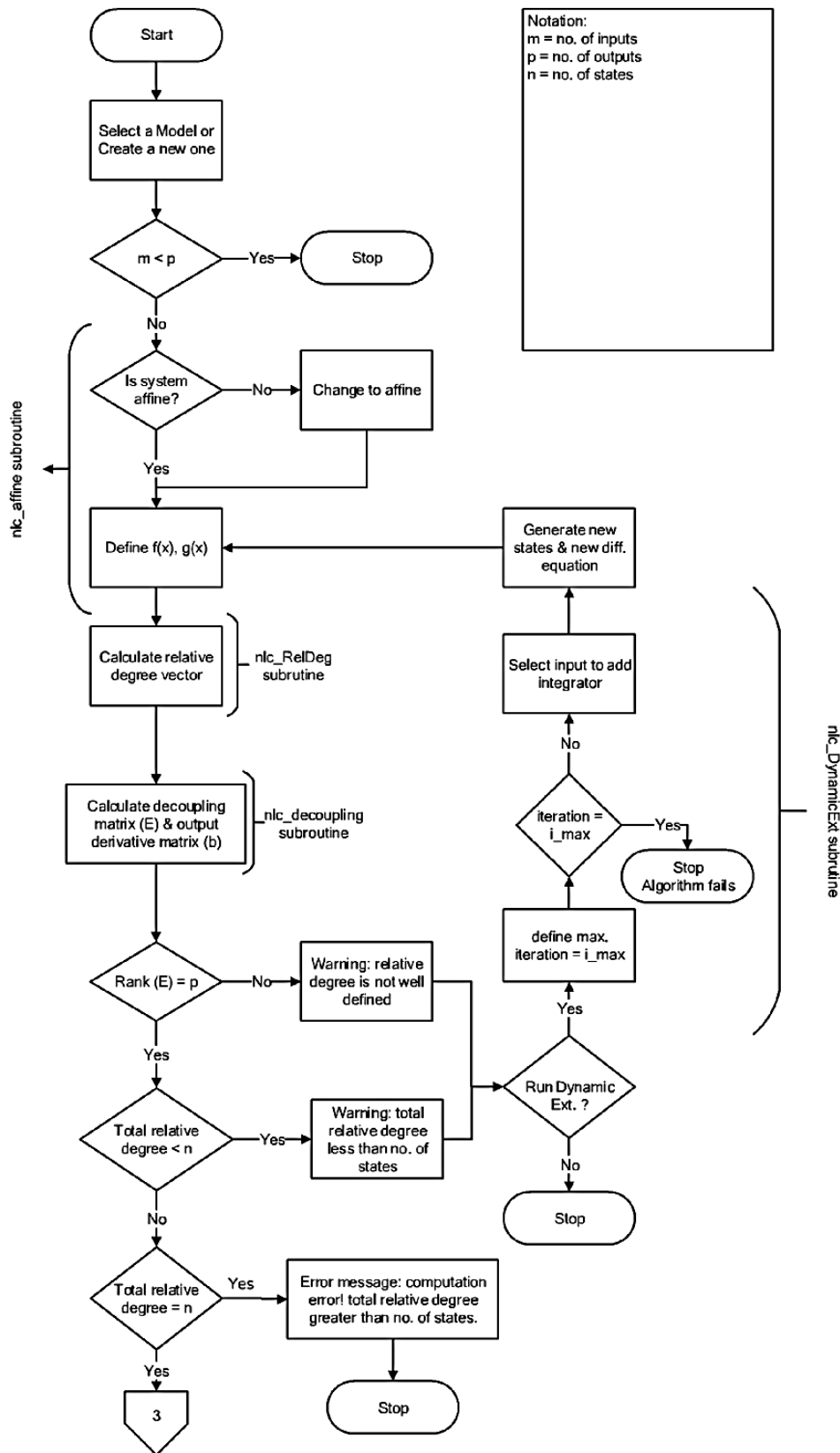
- [1] A. Isidori, A. Krener, C. Gori-Giorgi, and S. Monaco, "Nonlinear decoupling via feedback: A differential geometric approach," *IEEE Transactions on Automatic Control*, vol. 26, no. 2, pp. 331–345, 1981.
- [2] A. J. Krener, A. Isidori, and W. Respondek, "Partial and robust linearization by feedback," in *Proc. 22nd {IEEE} Conf. Decision Control*, 1983, pp. 126–130.
- [3] T. J. Tarn, A. K. Bejczy, A. Isidori, and Y. Chen, "Nonlinear feedback in robot arm control," in *The 23rd {IEEE} Conference on Decision and Control, 1984*, 1984, pp. 736–751.
- [4] R. Marino, "On the largest feedback linearizable subsystem," *Syst. Control Lett.*, vol. 6, no. 5, pp. 345–351, Jan. 1986.
- [5] P. J. McLellan, T. J. Harris, and D. W. Bacon, "Error trajectory descriptions of nonlinear controller designs," *Chem. Eng. Sci.*, vol. 45, no. 10, pp. 3017–3034, 1990.
- [6] C. I. Byrnes and A. Isidori, "Exact linearization and zero dynamics," in *29th IEEE Conference on Decision and Control (1990)*, 1990, pp. 2080–2084 vol.4.
- [7] C. Kravaris and M. Soroush, "Synthesis of multivariable nonlinear controllers by input/output linearization," *{AIChE} J.*, vol. 36, no. 2, pp. 249–264, Feb. 1990.
- [8] J.-J. E. Slotine, W. Li, and others, *Applied nonlinear control*, vol. 199, no. 1. Prentice-Hall Englewood Cliffs, {NJ}, 1991.
- [9] M. Soroush and C. Kravaris, "Nonlinear control of a batch polymerization reactor: An experimental study," *{AIChE} J.*, vol. 38, no. 9, pp. 1429–1448, Sep. 1992.
- [10] M. Guay and P. J. McLellan, "Input-output linearization of general nonlinear control systems," in *American Control Conference, 1994*, 1994, vol. 3, pp. 2695–2699 vol.3.
- [11] V. Polyakov, R. Ghanadan, and G. L. Blankenship, "Symbolic numerical computational tools for nonlinear and adaptive control," in *{IEEE}/{IFAC} Joint Symposium on Computer-Aided Control System Design, 1994. Proceedings*, 1994, pp. 117–122.
- [12] C. A. Schwartz, P. W. Gibbens, and M. Fu, "Achieving vector relative degree for nonlinear systems with parametric uncertainties," *Int. J. Robust Nonlinear Control*, vol. 5, no. 2, pp. 139–151, Jan. 1995.
- [13] K. B. McAuley, D. A. Macdonald, and P. J. McLellan, "Effects of operating conditions on stability of gas-phase polyethylene reactors," *{AIChE} J.*, vol. 41, no. 4, pp. 868–879, 1995.
- [14] A. 1995. N. control systems. B. N. Y. S. Isidori, *Nonlinear control systems*. Springer, 1995, p. 549.

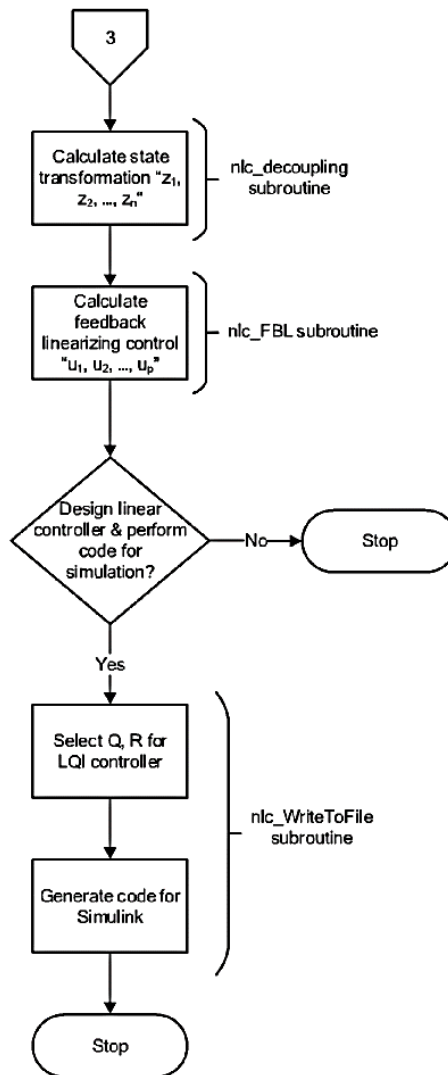
- [15] M. Soroush, “Nonlinear output-feedback control of a polymerization reactor,” in *American Control Conference, Proceedings of the 1995*, 1995, vol. 4, pp. 2672–2676 vol.4.
- [16] A. B. Ogunye, “{SYMCON}: Symbolic Computation Controller package for use with {MapleV},” in , *Proceedings of the 1996 {IEEE} International Symposium on Computer-Aided Control System Design, 1996*, 1996, pp. 488–494.
- [17] A. B. Ogunye, A. Penlidis, and P. M. Reilly, “Polynomial matrices for the design of multivariable control systems using symbolic computation,” in , *Proceedings of the Twenty-Eighth Southeastern Symposium on System Theory, 1996*, 1996, pp. 540–544.
- [18] M. Abdalla, R. Wang, and R. McLauchlan, “Solving Lyapunov equations symbolically,” in , *Proceedings of the 1996 {IEEE} International Symposium on Computer-Aided Control System Design, 1996*, 1996, pp. 504–509.
- [19] M. Guay, P. J. McLellan, and D. W. Bacon, “Computer algebra methods for feedback linearization using an exterior calculus framework,” in *American Control Conference, 1997. Proceedings of the 1997*, 1997, vol. 3, pp. 1385–1389 vol.3.
- [20] M. A. Henson and D. E. Seborg, Eds., *Nonlinear Process Control*. Upper Saddle River, {NJ}, {USA}: Prentice-Hall, Inc., 1997.
- [21] S. A. Dadebo, M. L. Bell, P. J. McLellan, and K. B. McAuley, “Temperature control of industrial gas phase polyethylene reactors,” *J. Process Control*, vol. 7, no. 2, pp. 83–95, 1997.
- [22] Z. Benyo, B. Palancz, C. Juhadz, and P. Varady, “Design of glucose control via symbolic computation,” in *Proceedings of the 20th Annual International Conference of the {IEEE} Engineering in Medicine and Biology Society, 1998*, 1998, vol. 6, pp. 3116–3119 vol.6.
- [23] “Polynomial Toolbox,” *PolyX, Ltd.*, 1998. [Online]. Available: <http://www.polyx.cz/>.
- [24] M. Soroush and N. Zambare, “Nonlinear output feedback control of a class of polymerization reactors,” *{IEEE} Trans. Control Syst. Technol.*, vol. 8, no. 2, pp. 310–320, Mar. 2000.
- [25] Simon G. Fabri and V. Kadiramanathan, *Functional Adaptive Control - An Intelligent Systems Approach*, 2001st ed. London: Springer-Verlag, 2001.
- [26] S. Kolavennu, S. Palanki, and J. C. Cockburn, “Nonlinear control of nonsquare multivariable systems,” *Chem. Eng. Sci.*, vol. 56, no. 6, pp. 2103–2110, Mar. 2001.
- [27] H. K. Khalil, *Nonlinear Systems*, 3 edition. Upper Saddle River, N.J: Prentice Hall, 2001.
- [28] G. C. Goodwin, “A brief overview of nonlinear control,” *Cent. Integr. Dyn. Control Bildir. Dep. Electr. Comput. Eng. Univ. Newcastle, Avustralya*, 2002.
- [29] K. Groves and A. Serrani, “Modeling and nonlinear control of a single-link flexible joint manipulator,” *APPENDICES*, 2004.
- [30] M. Ondera, “Matlab-Based Tools for Nonlinear Systems,” *Proc. 13th Annu. Conf. Tech. Comput.*, p. 96, 2005.

- [31] S. Ibrir, W. F. Xie *, and C.-Y. Su, “Observer-based control of discrete-time Lipschitzian nonlinear systems: application to one-link flexible joint robot,” *Int. J. Control*, vol. 78, no. 6, pp. 385–395, Apr. 2005.
- [32] *Symbolic Math Toolbox for Use with {MATLAB}: User’s Guide*. {MathWorks}, Incorporated, 2005.
- [33] S. A. Abramov, H. Q. Le, and Z. Li, “Univariate Ore Polynomial Rings in Computer Algebra,” *J. Math. Sci.*, vol. 131, no. 5, pp. 5885–5903, Dec. 2005.
- [34] N. M. Ghasem, “Design of a Fuzzy Logic Controller for Regulating the Temperature in Industrial Polyethylene Fluidized Bed Reactor,” *Chem. Eng. Res. Des.*, vol. 84, no. 2, pp. 97–106, Feb. 2006.
- [35] H. R. Karimi and M. R. J. Motlagh, “Robust Feedback Linearization Control for a non Linearizable {MIMO} Nonlinear System in the Presence of Model Uncertainties,” in *{IEEE} International Conference on Service Operations and Logistics, and Informatics, 2006. {SOLI} ’06*, 2006, pp. 965–970.
- [36] C. Panjapornpon, M. Soroush, and W. D. Seider, “Software for analytical nonlinear controller design,” in *American Control Conference, 2006*, 2006, p. 6 pp.–.
- [37] A. Witkowska, M. Tomera, and R. SMierzchalski, “A Backstepping Approach to Ship Course Control,” *Int. J. Appl. Math. Comput. Sci.*, vol. 17, no. 1, pp. 73–85, Mar. 2007.
- [38] G. Conte, C. H. Moog, and A. M. Perdon, *Algebraic Methods for Nonlinear Control Systems*. Springer Science & Business Media, 2007.
- [39] A. K. A. Bennoune, “Application of the dynamic linearization technique to the reduction of the energy consumption of induction motors,” *Appl. Math. Sci.*, no. 33, 2007.
- [40] A. Gani, P. Mhaskar, and P. D. Christofides, “Fault-tolerant control of a polyethylene reactor,” *J. Process Control*, vol. 17, no. 5, pp. 439–451, Jun. 2007.
- [41] M. A.-H. Ali, B. Betlem, G. Weickert, and B. Roffel, “Non-linear model based control of a propylene polymerization reactor,” *Chem. Eng. Process. Process Intensif.*, vol. 46, no. 6, pp. 554–564, Jun. 2007.
- [42] F. Chyzak, A. Quadrat, and D. Robertz, “OreModules: A symbolic package for the study of multidimensional linear systems,” in *Applications of time delay systems*, Springer, 2007, pp. 233–264.
- [43] M. T. Soylemez and I. Ustoglu, “Polynomial Control Systems [Product Review],” *{IEEE} Control Syst.*, vol. 27, no. 4, pp. 124–137, Aug. 2007.
- [44] G. Zhai and X. Xu, “A unified approach to analysis of switched linear descriptor systems under arbitrary switching,” in *Proceedings of the 48th {IEEE} Conference on Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. {CDC}/{CCC} 2009*, 2009, pp. 3897–3902.

- [45] F. Antritter and J. Middeke, “A toolbox for the analysis of linear systems with delays,” in *2011 50th {IEEE} Conference on Decision and Control and European Control Conference ({CDC}-{ECC})*, 2011, pp. 1950–1955.
- [46] F. Antritter and Johannes Middeke, “A toolbox for the analysis of linear systems with delays,” *Decis. Control Eur. Control Conf. (CDC-ECC), 2011 50th IEEE Conf.*, pp. 1950–1955, 2011.
- [47] K. Shojaei, A. Mohammad Shahri, and A. Tarakameh, “Adaptive feedback linearizing control of nonholonomic wheeled mobile robots in presence of parametric and nonparametric uncertainties,” *Robot. Comput. Integr. Manuf.*, vol. 27, no. 1, pp. 194–204, Feb. 2011.
- [48] M. Huba, S. Skogestad, M. Fikar, and M. Hovd, “Selected topics on constrained and nonlinear control,” *Slovakia, {ROSA}. Dolný Kubín*, 2011.
- [49] M. Brasel and P. Dworak, “Multivariable Adaptive Controller for the Nonlinear {MIMO} Model of a Container Ship,” *{TransNav}, Int. J. Mar. Navig. Saf. Sea Transp.*, vol. 8, no. 1, pp. 41–47, 2014.
- [50] R. Brockett, “The early days of geometric nonlinear control,” *Automatica*, vol. 50, no. 9, pp. 2203–2224, Sep. 2014.
- [51] G. Bellu, M. P. Saccomani, S. Audoly, and L. D’Angiò, “DAISY: A new software tool to test global identifiability of biological and physiological systems,” *Comput. Methods Programs Biomed.*, vol. 88, no. 1, pp. 52–61, 2007.

Appendix A: NLC Package Flow Chart





Appendix B: Program Codes of NLC Subroutines

B.1 Main Program "nlc_main.m"

```
% nlc_main.m - Main program file for NLC package (see description below).
%
% NLC: NonLinear Control package for design of nonlinear controllers
% based on MATLAB symbolic toolbox.
%
% The application carries out computations of relative degree, state
% transformation as well as input output nonlinear feedback according to
% the rules of exact linearization for SISO and MIMO systems. Conditions
% for MIMO systems is taht the number of inputs greater than or equal
% to outputs. This program accept an input of differential equations.
% A set of nonlinear models written in m files as ordinary differential
% equations are provided. Follow program instructions to see examples and/
% or to write your own model.
%
% To run NLC package type "nlc_main" in command line and hit "enter" key.
%
% (C) 2014-2015, Khalid E. Al-Khater, King Fahd University of Petrolume and
% Minerals (KFUPM), abumahdi1425@gmail.com
%

tic
clear all
clc

% Model Selection
disp(' ')
disp('*** Welcome to NLC MATLAB Package By Khaild Al-Khater, KFUPM *** ');
disp('Below is a list of nonlinear models where you can select to apply');
disp('nonlinear control design. You can enter your own model as well. ');
disp(' ');
disp('For Help and more Information about this package typ 0 "zero"');
disp(' ');
disp(' 1. Fluidized Bed Reactor (FBR) with 1st order heat exchanger model');
disp(' 2. Fluidized Bed Reactor (FBR) with staged heat exchanger model');
disp(' 3. Continous Stirred Tank Reactor (CSTR), Series Reaction (non-
square). ');
disp(' 4. Induction Motor');
disp(' 5. Semibatch Co-polymerization Reactor');
disp(' 6. User Defined Model');
disp(' 7. General nonlinear model examples');
disp(' ');
choices = input('Select a model by typing the corresponding number: ');
disp(' ')

if choices == 0
    help nlc_main
    break
elseif choices == 1
    disp(' ')
    disp('FBR with 1st order heat exchanger model is chosen. ');
    disp(' ')
end
```

B.1 Main Program "nlc_main.m"

```
[dx,x,u,h] = nlc_FBR_Model_1stOrder_HE;
elseif choices == 2
    disp(' ')
    disp('FBR with staged heat exchanger model is chosen. ');
    disp(' ')
    [dx,x,u,h] = nlc_FBR_Model_Staged_HE;
elseif choices == 3
    disp(' ')
    disp('CSTR, Series Reaction Model is selected. ');
    disp(' ')
    [dx,x,u,h] = nlc_CSTR_series;
elseif choices == 4
    disp(' ')
    disp('Induction Motor Model is selected. ');
    disp(' ')
    [dx,x,u,h] = nlc_Induction_Motor_Model;
elseif choices == 5
    disp(' ')
    disp('Semibatch Co-polymerization Reactor Model is selected. ');
    disp(' ')
    [dx,x,u,h] = nlc_Semibatch_Copoly_Reactor_Model;
elseif choices == 6
    disp('User defined model is selected. ');
    disp(' ')
    disp('After 10 seconds, you will be directed to m file editor where you can
type your own model');
    disp('follow the instruction and save the file when complete. ');
    disp('Come back here and press any key to continue...')
    pause(10)
    edit nlc_UserDefinedModel
    pause
    [dx,x,u,h] = nlc_UserDefinedModel;
elseif choices == 7
    disp(' ')
    [dx,x,u,h] = nlc_NonLinModel_example;
else
    error('Invalid Entry!!')
end

% Retrieve number of inputs and ouptus
p = length(h);
m = length(u);

% This package valid for only systems with m >= p
if m < p
    error('Number of inputs has to be greater than or equal to number of
outputs.')
end

% Exctract f(x) and g(x) (affine form)
[f,g,dx,x,u] = nlc_affine(dx,x,u);
disp(' ')
disp('States of the system are: '); disp(x)%pretty(x)
disp(' ');
disp('f(x) = '); disp(f)%pretty(f);
disp(' ');
```

B.1 Main Program "nlc_main.m"

```

disp('g(x) = '); disp(g)%pretty(g);
disp(' ');
disp('u = '); disp(u)%pretty(u);
disp(' ');
disp('h(x) = '); disp(h)%pretty(h);

% Compute Relative Degree
[r_vector,r_total] = nlc_RelDeg(f,g,h,x,u);

% Compute decoupling matrix and output derivatives vector
[E,b] = nlc_decoupling(f,g,h,x,r_vector,u);
disp(' ');
disp('Decoupling Matrix, E = ');disp(E)%pretty(E)
disp(' ');
n = length(dx);

% Disply the results along with system information
str0 = ['No. of States (n) is : ',num2str(n)];disp(str0);disp(' ');
str1 = ['Total relative degree (r_total) is : ',num2str(r_total)];disp(str1);disp(' ');
str2 = ['Relative degree vector (r_vector) is : ',num2str(r_vector),'];disp(str2);disp(' ');
str3 = ['Number of inputs (m) is : ',num2str(m)];disp(str3);disp(' ');
str4 = ['Number of outputs (p) is : ',num2str(p)];disp(str4);disp(' ');
str5 = ['Rank of decoupline matrix is : ',char(rank(E))];disp(str5);disp(' ');

% Analysis of the results
if rank(E) == p
    if (r_total < n)
        disp(' ');
        disp('WARNING: Total relative degree is lower than system order!!!');
        disp(' It is essential to analyze stability of internal
dynamics!');
        disp(' However, Applying Dynamic Extension Algorithm may help to
achieve full');
        disp(' order relative degree. ');
        disp('NOTE: Type "nlc_DynamicExt" to start Dynamic Extension
Algorithm. ');
        disp(' ');
    elseif (r_total == n)
        disp(' ');
        Z = nlc_StatTrans(f,h,x,r_vector); % State Transformation
        U = nlc_FBL(E,b); % Feedback linearizing control law
        disp(' ');
        disp('Following is the linearized state space system: ')
        [A,B,C,D] = nlc_EquLinSys(r_vector); % Linearized State Space System
        disp(' ');
        disp('Above A,B,C,D matrices can be used for linear controller desing.
This package')
        disp('can prepare a code to be used in a simulink block "MatlabFunction"
where you')
        disp('can perform simulation more effectively. The code will have an
emeded LQI')
        disp(' controller. To prepare the code type "nlc_WriteToFile")
    else
        disp(' ');

```


B.1 Main Program "nlc_main.m"

```
        disp('Computation Error!! Relative Degree becomes greater than number');
        disp('of states. Pleaser verify and try again.');
```

end

```
else
    disp('Attention!! The system dose not have a well defined relative degree.');
```

disp('You may consider to apply Dynamic Extension Algorithm.');

```
    disp('NOTE: Type "nlc_DynamicExt" to start Dynamic Extension Algorithm.');
```

end

```
disp('
                                     ');
toc
```

B.2 FBR Model with First Order Heat Exchanger

```

function [dx,x,u,h] = nlc_FBR_Model_1stOrder_HE

clear all

% Definition of symbols for the state variables, inputs, and parameters
syms M1 Y T I1 Qd
x = [M1, Y, T, I1, Qd];
syms FM1f Fc Fg Vg Vp Cv R Pv kp Tref Ea ac kd mw1 Bw MrCpr Cppol Cpf Tf
syms DeltaHR tau Fw Cpw AU Cpc CpIn FIf Tw RR kpT

% Algebraic equations:

bt = Vp*Cv*sqrt((M1+I1)*RR*T-Pv);
RM1 = M1*kp*exp(-Ea*(1/T-1/Tref)/R)*Y;
Cpg = M1*Cpf/(M1+I1)+I1*CpIn/(M1+I1);
Hf = FM1f*Cpf*(Tf-Tref)+FIf*CpIn*(Tf-Tref);
Hr = -DeltaHR*mw1*RM1;
Hpol = Cppol*(T-Tref)*RM1*mw1;
Gamma = AU*((Fg*Cpg)^(-1)+1/Fw*Cpw);
Tg0ss = (Tw*(1-exp(Gamma))-T*(1-Fg*Cpg/Fw*Cpw))/(Fg*Cpg/Fw*Cpw-exp(Gamma));

% Differential equations:

dI1 = (FIf-I1*bt/(M1+I1))/Vg;
dM1 = (FM1f-M1*bt/(M1+I1)-RM1)/Vg;
dY = Fc*ac-kd*Y-RM1*mw1*Y/Bw;
dT = (Hf-Qd+Hr-Hpol)/(Bw*Cppol+MrCpr);
dQd = (Fg*Cpg*(x(3)-Tg0ss)-Qd)/tau;

PR = kpT*Y*M1*mw1;    % Production Rate
Pm = M1 * R * T;     % Monomer Partial Pressure
P = (M1+I1)*RR*T;    % reactor pressure

dx = [dM1, dY, dT, dI1, dQd];

% U = [Fw FM1f Fc FIf Tw];
% H = [M1 I1 Y T Qd PR P Pm];
u = [Fc;Fw];
h = [M1;T];

```

B.3 FBR Model with Staged Heat Exchanger

```

function [dx,x,u,h] = nlc_FBR_Model_Staged_HE

clear all

% Definition of symbols for the state variables, inputs, and parameters
syms M1 I1 Y T Tw1 Tg1
syms Vg Vp Cv R RR Pv kp Tref Ea ac kd mw1 Bw MrCpr Cppol Cpf Tf Cpg
syms DeltaHR AU CpIn FM1f Fc Fg F1f Fw Mw Twi Cpw Tw1 Tg1 Mg

% Algebraic equations:
bt      = Vp * Cv * sqrt((M1+I1) * RR * T - Pv);
RM1     = M1 * kp * exp(-Ea/R*(1/T-1/Tref)) * Y;
Cpg     = M1/(M1 + I1) * Cpf + I1/(M1 + I1) * CpIn;
Hf      = FM1f * Cpf * ( Tf - Tref) + F1f * CpIn * (Tf - Tref);
Hg1     = Fg * (Tg1 - Tref) * Cpg;
Hg0     = (Fg + bt) * (T - Tf) * Cpg;
Hr      = -DeltaHR * mw1 * RM1;
Hpol    = Cppol * (T - Tref) * RM1 * mw1;

% Differential equations:

dI1     = (F1f - I1/(M1 + I1) * bt)/Vg;
dM1     = (FM1f - M1/(M1 + I1) * bt - RM1)/Vg;
dY      = Fc * ac - kd * Y - RM1 * mw1 * Y/ Bw;
dT      = (Hf + Hg1 - Hg0 + Hr - Hpol)/(MrCpr + Bw * Cppol);
dTw1    = Fw/Mw * (Twi - Tw1) - AU/(Mw * Cpw) * (Tw1 - Tg1);
dTg1    = Fg/Mg * (T - Tg1) + AU/(Mg * Cpg) * (Tw1 - Tg1);

dx = [dI1,dM1,dY,dT,dTw1,dTg1];

PR = kp * exp(-Ea/R*(1/T-1/Tref))*Y*M1*mw1;    % Production Rate
Pm = M1 * RR * T;                             % Monomer Partial Pressure
P = (M1+I1)*RR*T;                             % reactor pressure

x = [M1, I1, Y, T, Tw1, Tg1];

u = [Fc;Fw];
h = [T;M1];

```

B.4 CSTR Model

```
function [dx,x,u,h] = nlc_CSTR_series

clear all

n = 5;
syms u1 u2 u3 k1 k2 cBS cAS cDS F V
x = sym(zeros(1,n));
for countr = 1 : n
    eval(sprintf('syms x%d', countr));
    x(:,countr) = eval(sprintf('x%d',countr));
end
dx(1) = -k1*(x(1)*x(2)+ x(1)*cBS + x(2)*cAS) - k2*(x(1)*x(4)+x(1)*cDS + ...
    x(4)*cAS) - (F/V)*x(1) + u1*F/V;
dx(2) = -k1*(x(1)*x(2) + x(1)*cBS + x(2)*cAS) - (F/V)*x(2) + u2*F/V;
dx(3) = k1*(x(1)*x(2) + x(1)*cBS + x(2)*cAS) - (F/V)*x(3);
dx(4) = -k2*(x(1)*x(4) + x(1)*cDS + x(4)*cAS) - (F/V)*x(4) + u3*F/V;
dx(5) = k2*(x(1)*x(4) + x(1)*cDS + x(4)*cAS) - (F/V)*x(5);
u = [u1;u2;u3];
h = [x(3);x(5)];
```

B.5 Induction Motor Model

```
function [dx,x,u,h] = nlc_Induction_Motor_Model

% States
syms omega psia psib ia ib J Ls Lr M Rs Rr np TL
x = [omega psia psib ia ib];

% Inputs
syms ua ub
u = [ua;ub];

% Differential Equations
domega = (np*M/J*Lr)*(psia*ib-psib*ia)-(TL/J);
dpsia = -(Rr/Lr)*psia-np*omega*psib+(Rr/Lr)*M*ia;
dpsib = -(Rr/Lr)*psib-np*omega*psia+(Rr/Lr)*M*ib;
dia = (M*Rr*psia)/((Lr*Ls-M^2)*Lr)+(np*M*omega*psib)/(Lr*Ls-M^2) ...
      -(M^2*Rr+Lr^2*Rs)*ia/((Lr*Ls-M^2)*Lr)+(Lr*ua)/(Lr*Ls-M^2);
dib = (M*Rr*psib)/((Lr*Ls-M^2)*Lr)+(np*M*omega*psia)/(Lr*Ls-M^2) ...
      -(M^2*Rr+Lr^2*Rs)*ib/((Lr*Ls-M^2)*Lr)+(Lr*ub)/(Lr*Ls-M^2);

% Parameters
J = 0.06;
Ls = 0.47;
Lr = 0.47;
M = 0.44;
Rs = 8;
Rr = 3.6;
np = 2;
TL = 4;

% States Derviatives Vector
dx = eval([domega,dpsia,dpsib,dia,dib]);

% Outputs
h = [omega;psia^2+psib^2];
```

B.6 User Defined Model

```

function [dx,x,u,h] = nlc_UserDefinedModel
disp(' ');
%% User Defined Model:
% this function is part of NLC package (type help nlc_main for more information).
%
% (C) 2014-2015, Khalid E. Al-Khater, King Fahd University of Petroleum and
% Minerals (KFUPM), abumahdi1425@gmail.com
%
% Here you can enter you own model in a ordinary differential format. below
% is an example of entering the model:
%
% first of all, you need to define number of states, copy below line and
% change the assigned number to "n"
%
% n = 5;
%
% then you need to define symbols for inputs and states, you can use the
% common convention (u1, u2, u3,..x1, x2, x3,..) or use you own convention
% (F, R, mu,...,T, S,...), however, in either cases you need to save your
% states in a column vector called "x" and inputs in a column vector called
% "u". Here an example of declaring five states as (x1,x2,..,x5):
%
% x = sym(zeros(1,n));
% for countr = 1 : n
%     eval(sprintf('syms x%d', countr));
%     x(:,countr) = eval(sprintf('x%d',countr));
% end
%
% in above the variables x1,x2,..,xn already stored in coloumn vector x.
% another way to define states is shown by following example:
%
% syms T F Rd Y G1
% x = [T; F; Rd; Y; G1];
%
% you can define the inputs in the same way. for example:
%
% syms u1 u2
%
% parameters can defined as variables as well in the similar way. like the
% example below we have 15 parameters labled as a1, a2, .., a15 defined as
% follows:
%
% for countr = 1 : 15
%     eval(sprintf('syms a%d', countr-1));
%     a(:,countr) = eval(sprintf('a%d',countr-1));
% end
%
% The next step is to write your differential equation. The derivatives can
% are stored in vector dx, so dx1/dt is entered as dx(1) = ... and dx2/dt
% is entered as dx(2) = ...
% another way is to assign derivative to any varialbe name like dT, dF, dRd
% and then define derivatives as dx = [dT; dF; dRd; ... ]
% one way to writed differential exqaution is given below:
% dx(1) = x(2);
% dx(2) = a(1) + a(2)*x(2) + a(3)*x(2)^2 + (a(4) + a(5)*x(4)-sqrt(a(6) + ...
%     a(7)*x(4)))*x(3)^2;
% dx(3) = a(8) + a(9)*x(3) + (a(10)*sin(x(4)) + a(11))*x(3)^2+u1;

```

B.6 User Defined Model

```

% dx(4) = x(5);
% dx(5) = a(12) + a(13)*x(4) + a(14)*x(3)^2*sin(x(4)) + a(15)*x(5) + u2;
%
% another way shown by following:
% dT = F/R + u1;
% dF = T*u2 - Y;
% dx = [dT; dF];
%
% The last step is to define outputs and store them in store them column
% vector "h", example is shown below:
%
% h = [x(1); x(3)];
%
% type your differential equations below. When you finish save this file
% and return to command window to continue the analysis. You can edit
% whatever written below or delete everything and type your model.
%
%% Hilocapter Model
% disp('Hilocapter Model...');
% n = 5;
% syms u1 u2
% x = sym(zeros(1,n));
% for countr = 1 : n
%     eval(sprintf('syms x%d', countr));
%     x(:,countr) = eval(sprintf('x%d',countr));
% end
% for countr = 1 : 15
%     eval(sprintf('syms a%d', countr-1));
%     a(:,countr) = eval(sprintf('a%d',countr-1));
% end
% dx(1) = x(2);
% dx(2) = a(1) + a(2)*x(2) + a(3)*x(2)^2 + (a(4) + a(5)*x(4)-sqrt(a(6) + ...
%     a(7)*x(4)))*x(3)^2;
% dx(3) = a(8) + a(9)*x(3) + (a(10)*sin(x(4)) + a(11))*x(3)^2+u1;
% dx(4) = x(5);
% dx(5) = a(12) + a(13)*x(4) + a(14)*x(3)^2*sin(x(4)) + a(15)*x(5) + u2;
%
% u = [u1;u2];
% h = [x(1);x(4)];
%
%% Robot with flexible joint
disp('Robot with flexible joint...');
n = 4;
syms u F1 J1 M g l k Fm Jm
x = sym(zeros(1,n));
for countr = 1 : n
    eval(sprintf('syms x%d', countr));
    x(:,countr) = eval(sprintf('x%d',countr));
end
dx(1) = x2;
dx(2) = -(F1/J1)*x2-(M*g*l/J1)*sin(x1)-(k/J1)*(x1-x3);
dx(3) = x4;
dx(4) = -(Fm/Jm)*x4+(k/Jm)*(x1-x3)+(1/Jm)*u;

% Parameters
Jm = 3.7e-3;

```

B.6 User Defined Model

```
J1 = 9.3e-3;  
M = 2.1e-1;  
l = 3.1e-1;  
k = 1.8e-1;  
Fm = 4.6e-2;  
F1 = 3.0e-2;  
g = 9.8e-3;  
  
dx = eval(dx);  
  
h = x1;  
%
```


B.7 General Nonlinear Model Examples

```

function [dx,x,u,h] = nlc_NonLinModel_example

% Model Selection
disp(' ')
disp(' 1. An example of Dynamic Extension Algorithm Failure (MIMO).');
disp(' 2. An example of Dynamic Extension Algorithm Success (MIMO).');
disp(' 3. Mechanical Arm (SISO).');
disp(' 4. The Contrived Model (MIMO).');
disp(' 5. Go back to previous menu.');
```

choice = input('Select a model by typing corresponding serial number: ');

```

if choice == 1
    n = 4;
    syms u1 u2
    x = sym(zeros(1,n));
    for countr = 1 : n
        eval(sprintf('syms x%d', countr));
        x(:,countr) = eval(sprintf('x%d',countr));
    end
    dx(1) = x(1)+x(1)*x(4)+x(3)*u1+u2;
    dx(2) = x(2)*exp(x(3))+u1;
    dx(3) = x(2)+x(3)^2;
    dx(4) = x(1)+x(2)-x(4)+x(1)*x(4)+(1+x(3))*u1+u2;

    u = [u1;u2];
    h = [x(1);x(2)];
elseif choice == 2
    n = 3;
    syms u1 u2
    x = sym(zeros(1,n));
    for countr = 1 : n
        eval(sprintf('syms x%d', countr));
        x(:,countr) = eval(sprintf('x%d',countr));
    end
    dx(1) = u1*cos(x(3));
    dx(2) = u1*sin(x(3));
    dx(3) = u2;

    u = [u1;u2];
    h = [x(1);x(2)];
elseif choice == 3
    n = 4;
    syms u I1 m1 lc1 I2 m2 l1 lc2 m2 K
    x = sym(zeros(1,n));
    for countr = 1 : n
        eval(sprintf('syms x%d', countr));
        x(:,countr) = eval(sprintf('x%d',countr));
    end

    a = I1 + m1*lc1^2 + I2 + m2*(l1^2 + lc2^2);
    b = I2 + m2*lc2^2;
    c = m2*l1*lc2;
    d = b*(a - b);
    Delta = d - (cos(x(2)))^2;

```

B.7 General Nonlinear Model Examples

```
dx(1) = x(3);
dx(2) = x(4);
dx(3) = (b*c*(x(3) + x(4))^2*sin(x(2)) + (b + c*cos(x(2)))*K*x(2) + ...
        c^2*x(3)*sin(x(2))*cos(x(2)))/Delta + u*b/Delta;
dx(4) = -((b+c*cos(x(2)))*(x(4)+2*x(3))*c*x(4)*sin(x(2)) + (a + ...
        2*cos(x(2)))*(c*x(3)^2*sin(x(2)) + K*x(2)))/Delta - u*(b + ...
        c*cos(x(2)))/Delta;

h = x(1);
elseif choice == 4
    n = 5;
    syms u1 u2
    x = sym(zeros(1,n));
    for countr = 1 : n
        eval(sprintf('syms x%d', countr));
        x(:,countr) = eval(sprintf('x%d',countr));
    end
    dx(1) = x(2) + x(2)^2 + u2;
    dx(2) = x(3) - x(1)*x(4) + x(4)*x(5);
    dx(3) = x(2)*x(4) + x(1)*x(5) - x(5)^2 + u1*cos(x(1)-x(5)) + u2;
    dx(4) = x(5);
    dx(5) = x(2)^2 + u2;
    u = [u1;u2];
    h = [x(1) - x(5);x(4)];
elseif choice == 5
    dx = [];
    x = [];
    u = [];
    h = [];
    run nlc_main
else
    disp('Invalid Entry!!')
end
```

B.8 Affine check and affine conversion "nlc_affine.m" & "nlc_MakeAffine.m"

```
function [f,g,dx,x,u] = nlc_affine(dx,x,u)

n = length(dx);
m = length(u);
g = jacobian(dx,u);
f = vpa(zeros(n,1));
gvar = symvar(g);
affine = 1;
index = 0;
unaffine = [];
for i = 1:m
    if ~isempty(find(gvar == u(i)))
        affine = 0;
        index = index + 1;
        unaffine = [unaffine,find(u == u(i))];
    end
end
if affine == 1
    disp(' ');
    disp('System is affine!');
    for i = 1:n
        f(i,:) = dx(i) - g(i,:)*u;
    end
else
    disp(' ');
    disp('System is not affine!');
    disp(' ');
    disp('...Changing to affine by adding integrator(s) to the non-affine
input(s).');
    [f,g,dx,x,u] = nlc_MakeAffine(dx,x,u,unaffine);
end

function [f,g,dx,x,u] = nlc_MakeAffine(dxn,xn,un,unaffine)

NoOfInt = 0;
for i = 1:length(unaffine)
    ui = unaffine(i);
    n_xn = length(xn);
    xn(n_xn+1) = un(ui);
    NoOfInt = NoOfInt + 1;
    eval(sprintf('syms alpha%d', NoOfInt));
    un(ui) = eval(sprintf('alpha%d',NoOfInt));
    n_dxn = length(dxn);
    dxn(n_dxn+1) = un(ui);
end
[f,g] = nlc_affine(dxn,xn,un);
dx = dxn;
x = xn;
u = un;
```

B.9 Relative Degree Computation Program "nlc_RelDeg" and Lie Derivative "nlc_liederivative"

```

function [r_vector,r_total] = nlc_RelDeg(f,g,h,x,u)
p = length(h);
r = ones(1,p);
m = length(u);
LgLfh = vpa(zeros(1,p));
for i = 1:p
    flag = 0;
    k = 1;
    while flag == 0
        if k == 1
            r(i) = k;
            Lfh = h(i);
            for j=1:m
                LgLfh(j) = nlc_liederivative(g(:,j),h(i),x);
            end
        else
            r(i) = k;
            Lfh_new = nlc_liederivative(f,Lfh,x);
            for j=1:m
                LgLfh(j) = nlc_liederivative(g(:,j),Lfh_new,x);
                Lfh = Lfh_new;
            end
        end
    end
    for z=1:m
        if LgLfh(z) ~= 0
            flag = 1;
        end
    end
    if flag == 0
        k = k+1;
    end
end
end
r_vector = r;
r_total = sum(r);

function L = nlc_liederivative(F,h,x)
Pd_h = jacobian(h,x);
L = transpose(F)*transpose(Pd_h);
return

```

B.10 Decoupling and Output Derivatives Matrices "nlc_decoupling"

```
function [E,b] = nlc_decoupling(f,g,h,x,r,u)
p = length(h);
m = length(u);
LgLfh = vpa(zeros(1,p));
b = vpa(zeros(p,1));
E = vpa(zeros(p,m));
for i=1:p
    for k=1:r(i)
        if k == 1
            Lfh = h(i);
            for j=1:m
                LgLfh(j) = nlc_liederivative(g(:,j),h(i),x);
            end
        else
            Lfh_new = nlc_liederivative(f,Lfh,x);
            for j=1:m
                LgLfh(j) = nlc_liederivative(g(:,j),Lfh_new,x);
                Lfh = Lfh_new;
            end
        end
    end
    b(i) = nlc_liederivative(f,Lfh,x);
    E(i,:) = LgLfh;
end
```

B.11 State Transformation Program "nlc_StatTrans.m"

```
function Z = nlc_StatTrans(f,h,x,r_vector)

p = length(h);
n = length(f);
r = sum(r_vector);

for i=1:p
    for k=1:r_vector(i)
        if k == 1
            Lfh = h(i);
        else
            Lfh_new = nlc_liederivative(f,Lfh,x);
            Lfh = Lfh_new;
        end
        z(k,:) = Lfh;
    end
    if i == 1
        Z = z;
    else
        Z = [Z;z];
    end
    clear z
end
disp(' ');
disp('Transformation Equations, Z = ');
disp(' ');
for i=1:length(Z)
    zlen = length(char(Z(i)));
    if zlen > 1500
        str1 = '...WARNING!! The formula is too long and will not be
displayed';
        str2 = ['z',num2str(i),' = ',str1];
        disp(str2);
    else
        str = ['z',num2str(i),' = ',char(Z(i))];
        disp(str);
    end
end
end
```

B.12 Feedback Linearizing Control Law Computation "nlc_FBL.m"

```
function u = nlc_FBL(E,b)
% Feed Back Linearizing Controller
[p,m] = size(E);
v = sym(zeros(p,1));
for k = 1 : p
    eval(sprintf('syms v%d', k));
    v(k,:) = eval(sprintf('v%d', k));
end

disp(' ');
disp('Nonlinear feedback, U = ');
disp(' ');

% Control Law %
Evar = symvar(E);
Evar = sym(Evar, 'real');
if p == m
    Einv = E^-1;
    u = Einv*(v - b);
    for i=1:m
        ulen = length(char(u(i)));
        if ulen > 1500
            str1 = '...WARNING!! The formula is too long and will not be
displayed';
            str2 = ['u',num2str(i), ' = ',str1];
            disp(str2);
        else
            str = ['u',num2str(i), ' = ',char(u(i))];
            disp(str);
        end
    end
else
    for i=1:m
        Epinv = pinv(E);
        u = Epinv*(v - b);
        ulen = length(char(u(i)));
        if ulen > 1000
            str1 = '...WARNING!! The formula is too long and will not be
displayed';
            str2 = ['u',num2str(i), ' = ',str1];
            disp(str2);
        else
            str = ['u',num2str(i), ' = ',char(u(i))];
            disp(str);
        end
    end
end
end
```

B.13 Linearized State Space System " nlc_EquLinSys.m"

```
function [A,B,C,D] = nlc_EquLinSys(R)

% Linearized State Space System: This function is used to extract the equavilent
% linear system given the Relative Vector R.

Rt = sum(R);
p = length(R);
A = zeros(Rt,Rt);
B = zeros(Rt,p);
C = zeros(p,Rt);
i = 1;

% Matrix A
for j = 1:length(R)
    for ii = i:sum(R(1:j))
        if ii ~= sum(R(1:j))
            A(ii,ii+1) = 1;
        end
    end
    i = ii + 1;
end

% Matrix B
for i = 1:p
    B(sum(R(1:i)),i)=1;
end

% Matrix C
for i = 1:p
    if i == 1
        C(i,i) = 1;
    else
        C(i,sum(R(1:i-1))+1) = 1;
    end
end

% Matrix D
D = 0;

disp(' ')
disp('A = ');disp(A)%pretty(sym(A));
disp('B = ');disp(B)%pretty(sym(B));
disp('C = ');disp(C)%pretty(sym(C));
disp('D = ');disp(D)%pretty(sym(D));
```


B.14 Simulation Code Generator "nlc_WriteToFile.m"

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%nlc_WriteToFile: Generate a function for use with Simulink MATLABFunction
Block.%.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear mn unn nn xnn dxnn p Q R r q ql zd

% Retrieving size of the system
if exist('un','var') % Check if inputs are generated by DynamicExt.
    mn = length(un);
    unn = un; % store the inputs in a new variable.
    nn = length(dxn);
    dxnn = dxn; % store differential equations in a new variable.
    xnn = xn; % store states in a new variable.
else
    mn = length(u);
    unn = u;
    nn = length(dx);
    dxnn = dx;
    xnn = x;
end
p = length(h);

% Creating new file or overwiting if exist and writing first line to define
% the function for use with MATLABFunction - Simulink.
DiffEq = fopen('nlc_DiffEqFile.m','w');
str = 'function ';
fwrite(DiffEq,str);
fclose(DiffEq);
% Define states derivatives (function outputs)
DiffEq = fopen('nlc_DiffEqFile.m','a');
for i = 1:nn
    if i == 1
        clear str
        str = ['dx',num2str(i),','];
        fwrite(DiffEq,str);
    else
        clear str
        str = ['dx',num2str(i),','];
        fwrite(DiffEq,str);
    end
end
% Define error derivatives (function outputs)
for j = 1:p
    clear str
    str = ['dx',num2str(i+j),','];
    fwrite(DiffEq,str);
end
% Define system outputs (function outputs)
for j = 1:p
    if j == p
        clear str
        str = ['h',num2str(j),'] = '];
        fwrite(DiffEq,str);
    else
        clear str
        str = ['h',num2str(j),','];
    end
end

```

B.14 Simulation Code Generator "nlc_WriteToFile.m"

```
fwrite(DiffEq, str);
end
end
% Define states (function inputs)
for i = 1:nn
    if i == 1
        clear str
        str = ['nlc_DiffEqFile(', char(xnn(i)), ', '];
        fwrite(DiffEq, str);
    else
        clear str
        str = [char(xnn(i)), ', '];
        fwrite(DiffEq, str);
    end
end
% Define error (functions inputs)
for j = 1:p
    clear str
    str = ['x', num2str(i+j), ', '];
    fwrite(DiffEq, str);
end
% Define desired outputs (function inputs)
for j = 1:p
    if j == p
        clear str
        str = ['yd', num2str(j), ')'];
        fwrite(DiffEq, str);
    else
        clear str
        str = ['yd', num2str(j), ', '];
        fwrite(DiffEq, str);
    end
end
% State transformation (define z's)
for i = 1:length(Z)
    if i == 1
        clear str
        str = ['z = ', '[ ', char(Z(i)), ';'];
        fprintf(DiffEq, '\n %s', str);
    else
        clear str
        str = [char(Z(i)), ';'];
        fwrite(DiffEq, str);
    end
end
for j = 1:p
    if j == p
        clear str
        str = ['x', num2str(nn+j), ' ]'];
        fwrite(DiffEq, str);
    else
        clear str
        str = ['x', num2str(nn+j), ', '];
        fwrite(DiffEq, str);
    end
end
disp('')
```

B.14 Simulation Code Generator "nlc_WriteToFile.m"

```

disp('LQI controller design...')
disp(' ')
q = input('please specify the states weight, q =: ');
r = input('please specify the inputs gain, r =: ');
if m ~= p
    R = r*eye(p);
else
    R = r*eye(mn);
end
Q = q*C'*C;
Q1 = [Q;zeros(p,length(Z))];
Q2 = [zeros(length(Z),p);q*eye(p)];
Q3 = [Q1,Q2];
sys = ss(A,B,C,D);
[K,S,e] = lqi(sys,Q3,R);
clear str
[Ki,Kj] = size(K);
for i = 1:Ki
    for j = 1:Kj
        if i == 1 && j == 1
            clear str
            str = ['K = [ ',num2str(K(i,j))',' '];
            fprintf(DiffEq,'\n %s',str);
        elseif i ~= Ki && j == Kj
            clear str
            str = [num2str(K(i,j))',' '];
            fwrite(DiffEq,str);
        elseif i == Ki && j == Kj
            clear str
            str = [num2str(K(i,j))',' '];
            fwrite(DiffEq,str);
        else
            clear str
            str = [num2str(K(i,j))',' '];
            fwrite(DiffEq,str);
        end
    end
end
clear str
str = 'v = -K*z;';
fprintf(DiffEq,'\n %s',str);
% Writing linearizing control law
[Km,Kn] = size(K);
for i = 1:Km
    clear str
    str = ['v',num2str(i),' = ', 'v(',num2str(i),')',' '];
    fprintf(DiffEq,'\n %s',str);
end
for i = 1:length(U)
    clear str
    str = [char(unn(i)),' = ',char(U(i))',' '];
    fprintf(DiffEq,'\n %s',str);
end
% Writing differential equations and outputs
for i = 1:nn
    clear str
    str = ['dx',num2str(i),' = ',char(dxnn(i))',' '];

```

B.14 Simulation Code Generator "nlc_WriteToFile.m"

```
fprintf(DiffEq, '\n %s', str);
end
ii = i ;
for i = 1:p
    clear str
    str = ['h', num2str(i), ' = ', char(h(i)), ';'];
    fprintf(DiffEq, '\n %s', str);
end
for j = 1:p
    clear str
    str = ['dx', num2str(ii+j), ' = yd', num2str(j), ' - h', num2str(j), ';'];
    fprintf(DiffEq, '\n %s', str);
end
% closing and saving the file
fclose(DiffEq);
disp(' ')
disp('The code will be shown below, select and copy then past it in
MatlabFunction block')
disp('which is in the Simulink file. Once you copy press any key then the
Simulink file')
disp('will be opened. You can save the simulation outputs to workspace and name
them as:')
disp('y1,y2,.. for outputs and yd1, yd2,... for desired ouputs. After runing the
simulink')
disp('you can plot the resluts using the command "nlc_PlotSimulation"')
disp(' ')
disp('Press any key when your are ready...')
pause
disp(' ')
disp('%%%%%%%%% Begining of the code %%%%%%%%%')
type nlc_DiffEqFile
disp(' ')
disp('%%%%%%%%% End of the code %%%%%%%%%')
disp(' ')
disp('Copy above code and past it in MatlabFunction block. You can save');
disp('the simulation outputs to workspace and name them as:')
disp('y1,y2,.. for outputs and yd1, yd2,... for desired ouputs. After runing the
simulink')
disp('you can plot the resluts using the command "nlc_PlotSimulation"')
disp(' ')
disp('Press any key when your are ready...')
pause
disp('This may take while, be patient please....')
open nlc_SimulationFile
```

B.15 Dynamic Extension Program "nlc_DynamicExt.m"

```

tic
NoOfInt = 0;
flag = 1;
clear xn dxn fn gn un max ui
xn = x;
n = length(x);
dxn = dx;
fn = f;
gn = g;
un = u;
En = E;
i = 0;
disp('Type the maximum number of iteration to run dynamic extenstion');
disp(' ');
disp('NOTE: normally the maximum iteration is n (number of states). If you');
disp('choose big number, you may experience showness in program execution');
disp('and endup with a very complicated model with many input delays!!');
disp(' ');
disp('If you decide to choose it equal to n, leave it blank and just hit enter
key!')
imax = input(': ');
if isempty(imax)
    imax = n;
end
p = length(h);
while flag == 1
    ms = length(un);
    i = i + 1;
    if ms > 1 && p > 1
        for j = 1:ms
            Enonzero(j) = length(find(En(:,j)));
        end
        [max,ui]=max(Enonzero);
    elseif p == 1
        for j = 1:ms
            if E(j) ~= 0
                ui = j;
            end
        end
    else
        ui = 1;
    end
    str = ['.....Adding integrator to input number ',num2str(ui)];
    disp(str);
    n_xn = length(xn);
    xn(n_xn+1) = un(ui);
    NoOfInt = NoOfInt + 1;
    eval(sprintf('syms mu%d', NoOfInt));
    un(ui) = eval(sprintf('mu%d',NoOfInt));
    n_dxn = length(dxn);
    dxn(n_dxn+1) = un(ui);
    %[fn,gn] = nlc_affine(dxn,un);
    [fn,gn,dxn,xn,un] = nlc_affine(dxn,xn,un);
    [r_vectorn,r_totaln] = nlc_RelDeg(fn,gn,h,xn,un);
    [En,bn] = nlc_decoupling(fn,gn,h,xn,r_vectorn,un);
    n_dxn = length(xn);
    ms = length(un);

```

B.15 Dynamic Extension Program "nlc_DynamicExt.m"

```

disp('New States, xn = ');disp(xn)%pretty(xn);
disp('New Inputs, un = ');disp(un)%pretty(un);
disp('New Decoupling Matrix, En = ');disp(En)%pretty(En);
disp('
');
str0 = ['No. of States is : ',num2str(n_dxn)];disp(str0);disp(' ');
str1 = ['Total relative degree is : ',num2str(r_totaln)];disp(str1);disp('
');
str2 = ['Relative degree vector is :
',num2str(r_vectorn),']'];disp(str2);disp(' ');
str3 = ['Number of inputs and outputs is : ',num2str(ms)];disp(str3);disp('
');
str4 = ['Rank of decoupling matrix is : ',char(rank(En))];disp(str4);disp('
');
if rank(En) == ms && r_totaln == n_dxn
    flag = 0;
    disp('Full System Order Relative Degree Achieved!!')
    disp('
')
    disp('New f(x), fn = ');disp(fn)%pretty(fn);
    disp('
')
    disp('New g(x), gn = ');disp(gn)%pretty(gn);
    Z = nlc_StatTrans(fn,h,xn,r_vectorn); % State Transformation
    U = nlc_FBL(En,bn); % Feedback linearizing control law
    disp(' ');
    disp('Following is the linearized state space system: ')
    [A,B,C,D] = nlc_EquLinSys(r_vectorn); % Linearized State Space System
    disp('
');
    disp('Above A,B,C,D matrices can be used for linear controller desing.
This package')
    disp('can prepare a code to be used in a simulink block "MatlabFunction"
where you')
    disp('can perform simulation more effectively. The code will have an
emeded LQI')
    disp(' controller. To prepare the code type "nlc_WriteToFile"')
elseif i == imax
    flag = 0;
    disp('Reached to maximum number of iterations!! Relative Degree cannot be
achieved.')
```

end
clear ui max
end
toc

B.16 Plot Simulation Code "nlc_PlotSimulation.m"

```
for i = 1:p
    str = ['y', num2str(i)];
    figure(i)
    plot(eval(str));
    hold
    strd = ['yd', num2str(i)];
    plot(eval(strd), 'r');
    strt = ['Output ', '(' , num2str(i), ') '];
    title(strt)
    legend('Actual', 'Desired')
end
```

Vitae

Name :Khalid E. Al-Khater |

Nationality :Saudi |

Date of Birth :9/21/1978|

Email :abumahdi1425@gmail.com|

Address :Suadia Arabia, Qatif City.|

Academic Background : BS Degree from KFUPM, KSA, in Systems Engineering in 2002.
Participated in Process Control & Industrial Automation Summit in
The Netherlands with paper presentation "Application of SIS Life
Cycle in BMS systems".