INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600
A MODEL-BASED APPROACH TO VIBRATION ANALYSIS AND DIAGNOSIS

IHAB WATTAR

Bachelor of Science in Electrical Engineering

Southern Illinois University, Carbondale

December, 1987

Master of Science in Electrical Engineering

Southern Illinois University, Carbondale

December, 1989

Submitted in partial fulfillment of requirements for the degree

DOCTOR OF ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

May, 2000
This dissertation has been approved
for the Department of Electrical and Computer Engineering, Fenn College of Engineering
and the College of Graduate Studies by

[Signatures and dates]

Dissertation Committee Chairperson, Dr. Zhiqiang Gao, Date
Department of Electrical and Computer Engineering

Dr. James H. Burchart, Date
Department of Electrical and Computer Engineering

Dr. F. Eugenio Villaseca, Date
Department of Electrical and Computer Engineering

Dr. Jerzy T. Sawicki, Date
Department of Mechanical Engineering

Dr. James A. Locke, Date
Department of Physics
Dedication

To

My Parents for their Love,

Care, and Support

And

My wife Manal and my daughters Dalia and Dana for their

love and understanding
ACKNOWLEDGMENTS

I am grateful and thankful to my advisor Prof. Zhiqiang Gao for his time and effort he spent to help me define the dissertation topic. I would like also to express my deep appreciation to him, for his guidance, encouragement, kindness, support, and friendship. His invaluable advice and insightful thoughts were very helpful in making this work possible.

I would like also to thank Dr. J. Sawicki, Dr. J. Lock, Dr. F. Villaseca, Dr. Burghart, and Dr. W. Hafez, members of the dissertation committee, for their useful discussions, suggestions, comments, and advice. My special thanks to Dr. Sawicki for sharing his expertise about this topic.

My deepest, grateful thanks to Dr. W. Hafez for his help, guidance, hospitality, friendship, and encouragement, throughout the whole course of my work.

I will always be thankful to my parents for their enthusiastic encouragement, love, and support. I would like to give a special thanks to my dear wife Manal for her support and love that helped me through the rough times. She is extremely supportive of my goals and does her best to assist me. My deepest thanks to my sweet daughters Dalia and Dana, whom I love dearly, and was the major factor in my life to help me look forward to a brighter future.

I would like to thank all my family here in the US and in Syria for their support and encouragement.
I would like also to thank Dr. Suri (Spectra Quest Co.) for allowing me to use his simulator machine to collect real data for this research.

Special thanks to all my friends and co-workers for their enthusiastic encouragement during the course of this research.

Finally, I would like to thanks the R&D group at ABB for their support, with a special thanks to my supervisor, Mr. Dave Mielenik, and my coworker Joe Nemer for their support and encouragement.
A MODEL-BASED APPROACH TO VIBRATION ANALYSIS & DIAGNOSIS

IHAB WATTAR

ABSTRACT

This work addresses the problem of incipient fault detection and diagnostics for rotating machinery. This includes problem description, modeling of rotating machinery, fault mechanisms, and a fault diagnosis scheme. The proposed scheme is implemented in a simulation environment to test its feasibility. In this work, the faults (i.e. rub) of rotating machines are modeled in terms of nonlinear dynamic systems with quasi-periodic and chaotic behavior. It is shown that the non-smooth dynamic model of rub impact of rotating machines poses technical challenges to diagnosis algorithms that are based on nonlinear observers (such as, extended Kalman filter "EKF"). To avoid those challenges a regression viewpoint of the model was adopted, resulting in robust and effective algorithms for the monitoring and diagnosis of the rotor-stator process.

The regression-model approach is founded on computationally efficient algorithms for signal processing and parameter identification. A simulation study, which includes normal and different fault modes, illustrates the performance of the proposed approach, especially in the presence of measurement noise and process uncertainty.
Because of its robustness and simplicity, the proposed model-based approach is potentially of considerable value as a diagnostic tool in assessing condition-monitoring signals that are now routinely taken on modern rotating machinery.
TABLE OF CONTENTS

LIST OF TABLES........................................................................................................... x

LIST OF FIGURES.......................................................................................................... xi

CHAPTER

I. REVIEW AND OBJECTIVE .......................................................................................... 1
   1.1 Introduction ........................................................................................................... 1
   1.2 Main Results and Contribution ......................................................................... 4
   1.3 Organization of the Thesis ............................................................................... 6

II. RUB IMPACT PHENOMENA ..................................................................................... 8
   2.1 Introduction ........................................................................................................ 8
   2.2 Literature Review ............................................................................................ 9
   2.3 Mathematical Model Derivation ...................................................................... 12
   2.4 Simulation of the Rub Impact showing Sensitivity to Parameters ................. 18

III. SIGNAL PROCESSING AND DENOISING .............................................................. 25
    3.1 Introduction ...................................................................................................... 25
    3.2 Literature Review ........................................................................................... 26
    3.3 Wavelet Transforms VS. Fourier transform ....................................................... 32
    3.4 Denoising using wavelet ................................................................................ 36
       3.4.1 Definition of wavelet .............................................................................. 37
       3.4.2 Selection of wavelet parameters ............................................................. 38
       3.4.3 Denoising via threshold selection ............................................................ 40

IV. A MODEL-BASED APPROACH TO VIBRATION ANALYSIS AND
    DIAGNOSIS ........................................................................................................... 48
    4.1 Introduction ...................................................................................................... 48
    4.2 Problem Formulation ..................................................................................... 51
    4.3 Proposed Approach ....................................................................................... 55
       4.3.1 Off-line Parametric identification ......................................................... 55
       4.3.2 On-line Parameter Estimation .............................................................. 65

V. IMPLEMENTATION AND SIMULATION RESULTS ................................................. 70
    5.1 Introduction ...................................................................................................... 70
    5.2 Off-line Implementation and Simulation Results ........................................... 70

VI. EXPERIMENTAL RESULTS .................................................................................... 90
    6.1 Introduction ...................................................................................................... 90
6.2 Hardware Setup ....................................................................................... 92
6.3 Test Conducted ....................................................................................... 95

VII. CONCLUSION ............................................................................................. 104
  7.1 Concluding Remarks ............................................................................ 104
  7.2 Future Work ......................................................................................... 106

REFERENCES ...................................................................................................... Error! Bookmark not defined.

APPENDICES ..................................................................................................... 112
  A. Machine Specification ............................................................................ 113
  B. Spectral Analyzer Specification............................................................ 115
  C. Different Cases for the Experiment....................................................... 116
  D. Simulation Programs ............................................................................ 118


LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Fault categories and sample simulation parameter values</td>
<td>77</td>
</tr>
<tr>
<td>II. Difference between the actual and the estimated parameters</td>
<td>89</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Two D.O.F. Rotor-Stator System</td>
<td>13</td>
</tr>
<tr>
<td>2. Two D.O.F. Rotor-Stator System with Some of the Applied Forces in 3D</td>
<td>15</td>
</tr>
<tr>
<td>3. Free Body Diagram</td>
<td>16</td>
</tr>
<tr>
<td>4. History Diagram a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait</td>
<td>19</td>
</tr>
<tr>
<td>5. History Diagram with Noise a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait</td>
<td>20</td>
</tr>
<tr>
<td>6. History Diagram with Noise a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait, for a Chaotic System</td>
<td>21</td>
</tr>
<tr>
<td>7. History Diagram with Noise a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait, for a Chaotic System</td>
<td>22</td>
</tr>
<tr>
<td>8. History Diagram a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait, for a Transient Mode</td>
<td>23</td>
</tr>
<tr>
<td>9. Calculation of the Short Time Fourier Transform (upper view) and Instantaneous Spectral Density (lower view) of Vibration Signal</td>
<td>29</td>
</tr>
<tr>
<td>10. FFT for Case 12</td>
<td>32</td>
</tr>
<tr>
<td>11. Sinusoid with a Small Discontinuity</td>
<td>33</td>
</tr>
<tr>
<td>12. Comparison Between Wavelet and Fourier Transform for Discontinuity Detection</td>
<td>34</td>
</tr>
<tr>
<td>15. Chopping up the Signal into Appropriately Scaled and Shifted Wavelet</td>
<td>37</td>
</tr>
<tr>
<td>16. Wavelet Filtering Process</td>
<td>41</td>
</tr>
<tr>
<td>17. Denoising Approach Using Wavelets</td>
<td>43</td>
</tr>
<tr>
<td>18. State Trajectory for one of the Fault Cases (case 12). Showing Two Different Methods for Denoising, Using Wavelet Transform</td>
<td>44</td>
</tr>
<tr>
<td>19. State Trajectory for Another Fault Case (case 11). Showing Two Different Methods for Denoising, Using Wavelet Transform</td>
<td>45</td>
</tr>
<tr>
<td>20. Proposed Approach</td>
<td>55</td>
</tr>
<tr>
<td>21. Block Diagram for Gradient Descent Method</td>
<td>65</td>
</tr>
<tr>
<td>22. Block Diagram of the Proposed Model-Based Diagnosis Algorithm</td>
<td>69</td>
</tr>
<tr>
<td>23. Time History of x and y for a Period of 2π</td>
<td>71</td>
</tr>
<tr>
<td>24. Flowchart Diagram for Off-Line Diagnosis Algorithm</td>
<td>73</td>
</tr>
<tr>
<td>25. Value of $\zeta$ and $E$, Using Gradient Descends Approach</td>
<td>74</td>
</tr>
<tr>
<td>26. State Trajectory for Case 12, Where an Off-line Algorithm Was Used</td>
<td>75</td>
</tr>
<tr>
<td>27. Denoising and Parameters Estimation</td>
<td>76</td>
</tr>
<tr>
<td>28. Parameters for the Normal Case</td>
<td>78</td>
</tr>
<tr>
<td>29. (a) Original Signal (b) Simulated Model Signal</td>
<td>78</td>
</tr>
<tr>
<td>30. Parameters Convergence</td>
<td>79</td>
</tr>
<tr>
<td>31. Quasi Periodic Behavior of the System</td>
<td>79</td>
</tr>
<tr>
<td>32. Orbit Diagram of the Simulated Model</td>
<td>80</td>
</tr>
<tr>
<td>33. Parameters Convergence</td>
<td>80</td>
</tr>
<tr>
<td>34. Parameters Convergence</td>
<td>81</td>
</tr>
<tr>
<td>35. Parameters Convergence</td>
<td>81</td>
</tr>
<tr>
<td>36. State Trajectory (a) Additive Noise and (b) Simulated Model</td>
<td>82</td>
</tr>
<tr>
<td>37. Parameters Convergence in a Noisy Environment</td>
<td>83</td>
</tr>
<tr>
<td>38. Parameters Estimation Cont.</td>
<td>84</td>
</tr>
<tr>
<td>39. Block Diagram for Four Different Cases</td>
<td>85</td>
</tr>
<tr>
<td>Page</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>40</td>
<td>Parameter Estimates for a Sequence of Different Fault Modes.</td>
</tr>
<tr>
<td>41</td>
<td>State Trajectory / Orbit Diagram for Fault Mode [2000,4000]</td>
</tr>
<tr>
<td>42</td>
<td>Spectra Quest Machinery Simulator</td>
</tr>
<tr>
<td>43</td>
<td>Two Aluminum Rotors</td>
</tr>
<tr>
<td>44</td>
<td>Proximity Probes</td>
</tr>
<tr>
<td>45</td>
<td>Shaft Coupling Kit</td>
</tr>
<tr>
<td>46</td>
<td>Finite Element Model of Joint Location</td>
</tr>
<tr>
<td>47</td>
<td>Block Diagram for Parameters Estimation for Real Data</td>
</tr>
<tr>
<td>48</td>
<td>Block Diagram which has been Extracted from the Matlab to Show the Simulation Process</td>
</tr>
<tr>
<td>49</td>
<td>Normal Case at 84 Hz , a) Original Real Data, and  b) Simulated Data</td>
</tr>
<tr>
<td>50</td>
<td>Rub with a Rubber Pin (32 HZ). a) Original Real Data, and  b) Simulated Data</td>
</tr>
<tr>
<td>51</td>
<td>Rub with a Rubber Pin with an Imbalance (32 HZ). a) Original Real Data and b) Simulated Data</td>
</tr>
</tbody>
</table>
CHAPTER I
REVIEW AND OBJECTIVE

1.1 Introduction

In recent years, it has been shown that bounded, aperiodic, apparently chaotic vibration can occur in deterministic mathematical models with no random inputs or forces. The term “chaos” is generally used to distinguish such behavior from true random processes for which the input is random, and these chaotic motions have been called “strange attractors”. In the past, only periodic motions were sought, non-periodic behaviors were completely ignored and no reference had been made to the possibility of the existence of chaotic motion until recently.

Over the past twenty years, there has been a wide spread interest in the phenomena of chaos in rotating machines. Many researchers in the fields of mechanics, physics and chemistry have vigorously investigated these machines.

A nonlinear rotor dynamical problem such as rotor-stator, rub-impact, interaction that is of general interest as well as possibly leading to valuable diagnostic indicator is investigated in this dissertation. Although this system is well known to researchers in the field of rotor dynamics, their chaotic behavior has not been investigated fully. It is intended here to find results that can be useful in engineering
design, investigating and detecting the combinations of the system parameters leading to nonlinear behavior particularly chaos.

The analysis of the nonlinear effects in rotor-stator systems is extremely difficult, and this dissertation will present analytical procedures that will generate valid results over a wide range of parameters. Vibration problems involving small nonlinearities lend themselves to closed form solutions obtained by using conventional analytical techniques. These techniques (harmonic balance, multiple scales, the method of slowly varying parameters...etc.) usually assume that the system has a simple periodic response, which is then successively iterated upon convergence to an acceptable approximation to the actual response. However, these methods are limited and only applied to a restricted range of parameter values. Chaotic behavior is beyond the predictive capability of these methods. A solution procedure for strong nonlinear vibration problems is presented in this dissertation, where certain parameters of the system will be calculated off-line and on-line.

The optimum performance of many rotating machines is dependent on minimizing radial clearances between rotor and stator parts. Due to misalignment and mass unbalance, reducing clearances may lead to rotor rub against stationary parts causing complicated dynamic behavior that is often unpredictable.

In this dissertation, we attempt to discover some nonlinear diagnostic methods of rotating machines for practical application. Today, our industry is looking for a way to detect and identify problems in machines in their early stages. This ability is valuable because it allows for:

- Reduction of unscheduled down time
• Reduction of turnaround time

• Eliminating of periodic disassembly of a machine for the purpose of inspection

• Greatly reduced probability of a machine "crash"

According to the Power Research Institute (EPRI), turbine breakdown nationally costs the power generation industry, and eventually the consumer, an estimated $200 million a year. In addition, the cost of replacement of a complete turbo generator machine after a total, catastrophic failure ranges between $100 - $200 million dollars.
1.2 Main Results and Contribution

A new approach of modeling faults (i.e. rub) in rotating machines is presented in terms of nonlinear dynamic system with quasi-periodic and chaotic behavior. This work identifies a class of fault scenarios under which the well-accepted nonlinear observers and state filters (i.e. extended Kalman filter “EKF”) can’t be robustly used to monitor or diagnose the machinery. Specifically, nonlinear observers and state filtering approaches to the diagnosis of rotor dynamics, by Abdel-Magied [1, 2], are based on technical assumptions (namely, Lipschitzian dynamics – roughly, dynamics with bounded derivatives\(^1\)) that are not valid in the case of rotor dynamics (i.e., rub impact). The impact of improper modeling can not be underestimated especially in the case of chaotic rotor dynamics, due to the sensitivity to initial conditions, or the “butterfly effect”. The technical challenges (Lipschitzian smoothness requirement) apply as well to the recent approaches in chaotic dynamic predictions, i.e., global synchronization as described by Wang [3].

This work discusses different methods on how to detect incipient faults in rotating machinery from noisy vibration signals. It focuses on pattern recognition of fault signatures in non-stationary vibration signals. The following techniques were explored to address the problem of vibration monitoring and diagnosis:

\(^1\) The dynamics of the system \(\dot{x} = f(x,u,t)\) is said to be Lipschitzian if there exists a constant \(K < \infty\) such that for all \(\delta\), \(\|f(x) - f(x + \delta)\| \leq K\|\delta\|\). Intuitively, for small \(\delta\), the conditions is roughly that of bounded derivatives. The rub-impact dynamics does not satisfy this condition.
1. A new multi-resolution (multi-scale) DSP technique, based on wavelet transform, is proposed for the denoising of experimental vibration signals under certain signal-to-noise ratios.

2. A nonlinear dynamic system methodology is proposed to study experimental vibration data based on quantitative and qualitative behavior of rotating machines under fault conditions.

3. A model-based signal classification approach is proposed. In the proposed approach, faults (i.e. rub, imbalance) of rotating machines are modeled in terms of nonlinear dynamic system with quasi-periodic and chaotic behavior.

4. An off-line algorithm based on gradient descent method is presented.

5. An on-line algorithm based on parameter estimation and pattern recognition is presented.

The data used in this research was generated from a mathematical model and a hardware simulator of a rotating machine.

The main contribution of this dissertation is the formulation of the problem of fault detection and an on-line diagnosis of rotating machinery in a statistical model-based framework. This includes problem description, modeling of rotating machinery and faults mechanisms, formulation of the detection and diagnostics problem, and an implementation of the proposed scheme in a simulation environment to test the feasibility of this approach. More specifically, this model-based approach to diagnosis of nonlinear dynamics is based on parameter identification and pattern recognition.
The approach has been successfully applied to electromechanical systems, in the case of single faults, and compound faults, see [4] for details.

Unlike state estimators, parameter estimators are based on input-output models of the dynamics rather than the state information. Furthermore, nonlinear rotor dynamic models that are linear in the parameter can result in efficient on-line fault diagnosis algorithms. The method was successfully applied to a real simulator data, which was obtained from the Spectra Quest Co. The algorithm proved its robustness and was able to attenuate the noise without losing the significant information in the signal. The implementation of the proposed fault detection scheme has been done taking into consideration different fault scenarios. A simulation study, which includes normal and different fault modes, illustrates the potential of the proposed approach, especially in the presence of measurement noise and process uncertainty [4].

1.3 Organization of the Thesis

The explanation of the mechanical fault phenomena of interest “Rub Impact” and the derivation of the mathematical model are introduced in Chapter II. The Denoising approach and a comparison between different denoising methods are presented in Chapter III.

The formulation of the fault detection and diagnosis problem for rotating machinery in a statistical model-based framework and a discussion of the system design requirements are presented in Chapter IV.
The implementation issues, simulation, and performance results are discussed in Chapter V. Hardware simulation and the results in "real data" are shown in Chapter VI. Finally, conclusion and future work are summarized in Chapter VII.
CHAPTER II
RUB IMPACT PHENOMENA

2.1 Introduction

“Rub Impact” is the phenomena of intermittent or continuous contact between rotating and stationary parts at close-running clearance locations in a rotating machine. Rotating machinery operates most efficiently when the close sealing clearances between the rotor and the stator are kept to a minimum. Misalignments and rotor mass unbalance can readily lead to an operating condition where the rotor rubs against the stationary parts. These intermittent rubs can cause complicated nonlinear dynamic behavior that is difficult to predict or diagnose using conventional time-based analysis or spectrum analysis techniques. The persistent rubbing can lead to machine failure or at least accelerated the wearing of the close-clearance parts.

This chapter includes a literature review for modeling the rub impact phenomena, mathematical modeling of the phenomena, and a simulation of the model under different sets of parameters to illustrate the sensitivity of the model dynamic to variation in the model parameters.
2.2 Literature Review

Many investigators have studied nonlinear rotor dynamical systems involving bearing clearances. Kascack [5], developed a method for the direct integration of a rotor dynamic system experiencing rotor rub induced by blade loss. He found that both the blade loss and the rub impact phenomena generated high frequency vibration components.

Kraker and coworkers [6], studied periodic motion of a Laval rotor model with a nonlinear finite boundary stiffness and Coulomb friction. They showed that in the spectrum of the periodic solutions, higher harmonics are found in addition to the first harmonic calculated by the linear analysis.

Geradin and Kill [7] investigated the nonlinear dynamics of flexible rotors using approximate techniques such as equivalent linearization and transient analysis. Nonlinear interaction forces (rubbing, dead band clearance) were found to induce sub-harmonic resonances as well as amplitude jumps.

Neilson and Barr [8] discussed the spectral features of the response of a rigid rotor mounted in discontinuously nonlinear supports. In [9], Neilson and Barr considered the response of two elastically supported rotors, which run at different speeds having a bearing clearance effect at a common bearing support. Over certain speed ranges, the results revealed what they called fan-like spectral sidebands, which diverge with shaft speed. Similar side bands were investigated by Ehrich [10] and were referred to as spontaneous side banding. He noted that side band spacing
frequency is a whole number fraction \((1/J)\) of the operation speed, where \(J\) was called the repetition index.

Choy, Padovan and Li [11], studied the onsets, transient and steady phases of rub bounce events using an analytical modeling and signature analysis procedure to simulate the nonlinear dynamical characteristics associated with rotor-blade-seal-casing rub interactions. Muszynska [12] gave a mathematical and experimental description of partial rotor-to-stator rubs. The rotor response showed the existence of steady state subharmonic vibrations of the order \(1/2, 1/3, 1/4\ldots\) as a result of rotor transient free lateral vibrations following the impacts. Beatty [13] has simulated a two-degree-of-freedom rotor, which is made to orbitally vibrate through a "dead band" clearance terminated by a linear radially isotropic support stiffness, similar to a bilinear support. The contact was assumed to occur at a specified location on the housing. Bently [14] used a simple horizontal rotor model with a bearing clearance to explain the occurrence of subharmonics in his experimental results. Childs [15] used a perturbation technique to study the occurrence of subharmonics, assuming small nonlinearity for the bearing clearance. Simulations revealing aperiodic whirling motion were reported by Childs [16]. Saito [17] utilized a harmonic balance method (HBM) along with a fast Fourier transform (FFT) procedure to explain some nonlinear characteristics in a Jeffcott rotor on nonlinear supports. Day [18] and Zalik [19] proposed an interpretation involving a nonlinear natural frequency to explain the occurrence of aperiodic motion.

Abu-Mahfouz [20], Adams and Abu-Mahfouz [21], presented results for the forced dynamic behavior of a simple two degree of freedom model consisting of a rub
interaction between a rotor and a boundary with a nonlinear restoring force and showed the existence of chaotic vibration behavior.

There are only a few papers dealing with the problem of chaotic dynamics of rotor-casing rub interaction. Szczgielski and Schweitzer [22] studied the motion of a gyro-pendulum touching a rigid plane wall, which represents a vertically hung rotor. Solutions were classified by changing the impact restitution coefficient and rubbing friction coefficient. Their results show periodic, quasi-periodic and chaotic solutions, containing cascades of period doubling and Feigenbaum structures. Choy, Padovan and Yu [23], investigated the different regimes of rubbing, i.e. the development of full rubs, rigid bouncing and quasi chaotic behavior due to the changes in rotor-to-casing mass and stiffness ratios, as well as blade stiffness and friction effects. Using a numerical model of a rotor which simulates local contact with a stator in close proximity of a bilinear spring, Ehrich [24] and [25] studied a highly nonlinear simple single-mass-Jeffcott rotor system and found chaotic response in transition zones between successive orders of rub harmonic and super harmonic operation. Choi and Noah [26] and Kim and Noah [27] also used the HBM with FFT to show the occurrence of superharmonics, subharmonics, and possibly, in [27] chaos in a rotor in presence of bearing clearance.

Finally, Horattas [28], Horattas, Adams and Abdel-Magied [29] presented laboratory experiments that confirm and further explore earlier computer simulations related to rub-impact and chaos tracking techniques (i.e. Poincare’ maps). The results showed that rub-impact generated a chaotic vibration motion.
2.3 Mathematical Model Derivation

One of the classic methods in nonlinear dynamics is based on mathematical models (nonlinear differential equations), by means of which it is possible to determine the relationships among some parameters and dynamic behavior of the mechanical system. However, in the machinery diagnostics, simple differential equations can hardly characterize the dynamic behavior of complex machine set. The quantitative conclusions of such an analysis still remain very far from the practical results. Certainly, we do not rule out the possibility to use such results to explain some phenomena in practice.

To investigate the dynamics of rotor-casing systems during rub interaction, a simplistic rotor-stator model is employed in this study. The rub impact model consists of a de Laval rotor and nonlinear finite boundary stiffness with Coulomb friction as shown in Fig. 1.

The following is the list of variables and parameters associated with the rub impact model:

- \( K_s \): Isotropic stiffness of the shaft / bearing support
- \( D_s \): Isotropic damping of the shaft / bearing support
- \( F_R \): Normal component of the contact force
- \( F_T \): Tangential component of the contact force
- \( G \): Gravity
- \( \Omega \): Rotational speed
- \( (X, Y) \): Disc center coordinates with respect to the coordinates center
- \( \rho \): Radius
$\beta$: Mass center of the disk

$O^0$: Origin of the fixed coordinate system

$O$: Center of the disk

$M$: Mass of the disk

$t$: Time

Fig. 1 Two D.O.F. Rotor-Stator System
The Laval rotor is modeled as a mass-less shaft mounted in two bearings at each end. A thin rigid disc (the rotor) is located symmetrically between the bearings. The disc of mass $M$ rotates with a constant angular velocity $\omega$ with radius $\rho$ around the center $O$ of the disc. It is not necessary that the center of mass $\beta$ coincides with the center of the rotation $O$ as shown in Fig.1. The transverse bending vibrations of the rotor are modeled with two generalized coordinates. The disc translates in the $X$ and $Y$ directions and rotates around $Z$. The position of $O$ with respect to the origin of the coordinate system $O_0$ is given by the vector $r_0$ and the position of $\beta$ with respect to $O$ by $r_\beta$.

This mathematical 2-degree of freedom (D.O.F) Rotor-Stator model, which is shown in Fig. 1, was developed in [20] by Abu-Mahfouz to study the forced chaotic dynamical behavior of rotating machines. Specifically, it accounts for the rub interaction between a rotor and a boundary with a nonlinear restoring force.

Different programs in Fortran have been written by Abu-Mahfouz to represent the 18 different fault cases. In this thesis, Matlab has been used to regenerate the 18 different cases using the same mathematical model.

Fig. 2 and Fig. 3 show all different forces, which will affect the performance of the system. The disc is loaded with a constant gravitational force $G$ due to the weight of the disc. A linear restoring force "$K\dot{r}$" and "$D\dot{r}$" results from the isotropic stiffness $K_s$, and the isotropic damping $D_s$ of the shaft and its bearings.
The nonlinear force is produced by the contact and rubbing of the rotor against the stationary housing. The housing acts on the disc with a normal force and the tangential or friction force. The normal force is assumed to depend exponentially on the orbital radius of the shaft (Hertzian contact). It has been assumed that the relative velocity of the rotor contact point with respect to the housing contact point is always counterclockwise and not equal to zero. When rubbing between rotor and stator occurs, a tangential force proportional to the radial contact force and opposed to the surface velocity of the rotor is induced. The summation of all forces can be formulated using Newton's law.
Using Newton's Law and the model for contact forces (Hertzian), the differential equations describing the rub interaction behavior can be described as

\[
M \ddot{r}_m = M \left( \dot{r}_0 + \dot{r}_p \right) = \vec{G} + \vec{R} + \vec{F}_n
\]  

(2.3.1)
$M$: rotor mass

$\overline{G}$: Gravity

$\overline{R}$: Linear restoring force (bearings)

$\overline{F}_n$: Nonlinear contact and rubbing forces

$r_m$: Relative position of the mass center w.r.t. stator's center

$r_o$: Relative position of the rotor's center w.r.t. stator's center

$r_p$: Relative position of the mass center w.r.t. rotor's center

The above equation of motion and the rotor coordinate variables $X$ and $Y$ can be scaled with respect to the clearance, $C$, and the natural frequency $\omega_s$, as in [20]:

$$\ddot{x} + 2\zeta \dot{x} + x = -[K_\theta \delta^\alpha (\cos \theta - \mu \sin \theta)] + E \omega^2 \cos \omega t$$  \hspace{1cm} (2.3.2)

$$\ddot{y} + 2\zeta \dot{y} + y = -[K_\theta \delta^\alpha (\sin \theta + \mu \cos \theta)] + E \omega^3 \sin \omega t - G$$  \hspace{1cm} (2.3.3)

where $x = X/C: (-1 \leq x \leq +1)$ and $y = Y/C: (-1 \leq y \leq +1)$ are the scaled rotor coordinates (unless otherwise stated all references to rotor coordinates will refer to the scaled coordinates). The parameters of the scaled equation of motion are then defined as follow:

$\alpha$: Deformation exponent $1 \leq \alpha \leq 2$ (Hertzian restoring force)

$\theta$: $\tan^{-1} (y/x)$

$\xi$: Dimensionless damping

$\Omega$: Rotational speed.

$\omega_s$: System natural frequency, $\omega_s = \sqrt{\frac{K_s}{M}}$

$\omega$: Dimensionless rotational speed, $\omega = \frac{\Omega}{\omega_s}$

$\mu$: Coefficient of coulomb friction
E : Eccentricity ratio,  \( E = \frac{\varepsilon}{C} \), where C is the clearance, and \( \varepsilon \) is the distance between O and \( \beta \).

\( \delta \) : Contact condition (deformation)

\( K_\beta \) : Stiffness constant for the Hertzian contact force

\( \tau \) : Dimensionless time (\( \tau = \omega_s t \)) where \( \omega_s \) is the system natural frequency, and "t" is the time.

\[ G^* = \frac{G}{MC\omega_s^2} \]

C: Bearing mean radial clearance.

2.4 Simulation of the Rub Impact showing Sensitivity to Parameters

To show the effect of changes in the system parameters on the dynamics of the system, a mathematical simulation has been developed for the model given above. Hence, by choosing different parameters in equation (2.3.2) and (2.3.3), various failure scenarios can be simulated. An attempt by Abu-Mahfouz [20] has been made to simulate the complete global response of the system during rub events. The intention is to develop a basic understanding of the dynamics showing sub-harmonic, chaotic as well as quasi-periodic behavior of the system.

As an example, one of the fault cases with the following parameters was chosen (Case 12):

\[ [\omega \ E \ \mu \ \xi \ K_\beta \ G^* ] = [1.0 \ 0.49 \ 0.5 \ 0.14 \ 1.33e3 \ 0.13] \]

see the previous section for the definition of these parameters.
The state trajectory of the above case (Case 12), shows quasi-periodic motions (with respect to the stroboscopic Poincare’ map) of the rotor, with rubs all around the clearance circle. The Poincare’ map is simply a stroboscopic phase plane plot that only records those phase points at instant of time synchronous with the forcing period. Then the resulting set of points is projected into the displacement-velocity plane $(x, \dot{x})$. In our example the Poincare’ section shows that the attracting set is closed curve indicating quasi-periodic motion at two or more incommensurate frequencies, as shown in Fig. 4. By simulating equation (2.3.2) and (2.3.3), we obtained the trajectory data $x$ and $y$ (rotor dynamics with rub), which are plotted as follow:

Fig. 4 History Diagram a) $x$ vs. Time. b) $y$ vs. Time. c) State Trajectory. d) Phase Portrait.
Furthermore, by adding random noise to the signals x and y, with a signal to noise ratio of 20 dB, the state trajectory and the phase plot look drastically different, as shown below:

Fig. 5 History Diagram with Noise  a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait.
Again, by simulating equation (2.3.2) and (2.3.3) for different parameters:

\[
[\omega \ E \ \mu \ \xi \ K_\beta \ G ] = [ 1.0 \ 0.39 \ 0.1 \ 0.027 \ 4.0e3 \ 0.392 ]
\]

we obtained the trajectory data \( x \) and \( y \), which are plotted as follows (case 17 in [20]

by Abu-Mahfouz):

---

Fig. 6 History Diagram with Noise
a) \( x \) vs. Time. b) \( y \) vs. Time. c) State Trajectory. d) Phase Portrait, for a Chaotic System.

As shown in Fig. 6, the state trajectory of the system has a chaotic behavior with rubbing at many different points in its free flight within the clearance circle. In
addition, the poincare’ map shows a non-closed loop, which is an indication of the chaotic behavior of the system. With an additive noise, we obtain:

![Fig. 7 History Diagram with Noise a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait, for a Chaotic System.](image)

Again, with a noisy environment, i.e. due to sensor noise, the system trajectory might not give the right fault condition. In Chapter III, different methods will be presented to show how we can eliminate those unwanted noises without
loosing the original signal. In other words, removing the noise without compromising the high-frequency information.

If the steady state solution is not fully reached, transients can play a misleading role, as shown in Fig. 8, with the following parameters:

$$[\omega \ E \ \mu \ \xi \ K_B \ G] = [1.0 \ 1.0 \ 0.13 \ 0.11 \ 4.0e3 \ 0.392]$$

Fig. 8 History Diagram a) x vs. Time. b) y vs. Time. c) State Trajectory. d) Phase Portrait, for a Transient Mode.
Here, the Poincare' points of the transient solution take a shape similar to that of the chaotic attractor in Fig. 8-d. Although this transient, shown in Fig. 8-c, converges to a period-one steady state response, it is preceded by an extremely long chaotic transient that lasts for the rest of the iteration.

Considerable work has been done in the area of diagnostics in the last two decades. However, one might wonder, from all the above figures, how an engineer could monitor and recover different types of faults from noisy data. The following chapters will show different types of techniques, which were adopted for fault diagnosis, and different methods to eliminate the noise.
CHAPTER III
SIGNAL PROCESSING AND DENOISING

3.1 Introduction

Studying the characteristics of vibration signals of rotor dynamics (periodic, quasi-periodic, and chaotic) is essential for today's manufacturing and production industries. Engineers often have to diagnose and classify the nature of the unwanted oscillation components, in order to provide a clue on how to control them. Significance of diagnosis of vibration signals (Predictive maintenance) can yield considerable benefits to industry.

Signal processing has long been used in monitoring and fault diagnosis of rotating machinery, and it is also used in filtering noisy vibration signals. Techniques might vary, but each method might have many fascinating and attractive results. Some of these techniques are Fourier spectrum analysis, singular spectrum analysis, Wigner-Ville's method, fuzzy logic, Wavelets transform, and neural networks. Although spectral analysis has been the most common and widely used method in the rotating machine industries, it does not provide all the insight and clues needed to identify the cause of the vibration problem. Different types of signal processing
techniques for extracting fault signals are reviewed in the next section, followed by the introduction of Wavelet transform and its application in denoising.

3.2 Literature Review

For many years, analog filters were used to process the sound and vibration signals of a piece of machinery by either having banks of filters side by side and reading the output of each or by using a tunable filter to sweep through the frequency range of interest. Unfortunately, analog filters require relatively long settling time before accurate amplitude can be measured. In addition, sweeping through a frequency range was so slow that the machine was often under a different load condition at the end of a sweep than it was at the beginning.

In an attempt to speed up the process of gathering and processing data, constant percent filters, such as Octave or 1/3 Octave filters, were employed by Goldman [30]. Since a constant percent filter has less and less frequency resolution as frequency increases, an accurate description of machine characteristics was impossible.

Ever since Baron J.B. Fourier, the French mathematician showed that it is possible to represent any time waveform by a series of sine and cosine functions of particular frequencies and amplitudes, engineers have sought ways to simplify their work by looking for the frequency content of signals. In the machinery world, the goal is to relate the various frequencies seen in the spectra to the various physical phenomena occurring in a machine.
In 1965, Cooley and Tukey [31], of Columbia University, devised a fast Fourier transform (FFT) algorithm, making the calculation of the frequency spectra of a signal far more efficient and rapid than was previously possible in a digital computer. The commercial spectrum analyzer, capable of rapidly calculating the Fourier transform of a signal, quickly provides the frequency spectrum of a signal that would require hundreds of nondrifting, constant-frequency-width analog filters side by side. Various techniques based on the variation of FFT were introduced, including:

a) Power spectrum, or average spectrum, which was the average of the squared magnitude of a number of successive instantaneous spectra.

b) Power spectral density (PSD) of spectra, where the analyzer simply divides the amplitude of the measured power spectra at each frequency by the bandwidth of each frequency in hertz.

c) Average cross spectrum, which has been used in most basic of all dual-channel functions. It is used to retain the phase shift information between two channels (i.e. Ch. A and Ch. B). It is the complex product of spectrum Sb and complex conjugate of Sa.

d) Transmissibility, which is the real ratio of the output power spectrum to the input power spectrum. Transmissibility is fundamentally different from transfer function in that there is no phase information.

e) Transfer Function, where it is used to determine the natural frequencies of a structure, mode shapes, nodes, antinodes, and structural damping, and is the basis of modal analysis and operating deflection analysis.
f) Coherence, which is considered the most valuable dual-channel FFT function besides transfer function. Because its ability to measure the "goodness" of a transfer function test, as well as of its ability to help answer the question "who is doing what?". It measures the linear relationship between two channels.

g) Coherent output power, which is the power of signal \( B(t) \) that was caused by the power of signal \( A(t) \) or the power of signal \( B(t) \) that is being excited by the same phenomena that is exciting \( A(t) \), depending on whether one is dealing with a linear cause/effect relationship or not.

h) Impulse response, which is the inverse FFT of the transfer function. The advantage of solving input/output problems using impulse response is the plot of amplitude over time, which gives an intuitive picture of how a system reacts over time to an impulse.

i) Inverse transfer function, which deals with the problem of obtaining an incorrect value of transfer function in the frequency range near a natural frequency caused by instrumentation noise. It is the output power/cross spectrum.

j) Autocorrelation, which compares the similarity of a time domain function to replications of itself, delayed by a time constant \( \tau \). The main use of the Autocorrelation function is to check a signal for periodicity.

k) Cross-Correlation, which is very similar to Autocorrelation, except it deals with a 2-channel time domain property. It calculates the spectrum as an
inverse FFT. The cross-correlation function indicates the similarity of A(t) and B(t).

1) Cepstrum, the inverse FFT of the log of the complex spectrum.

Different techniques were also introduced to enhance the above conventional spectral analysis in order to handle non-stationary signals. One of those techniques is the so-called "Short-time Fourier transform" (STFT) (Newland 1993). It has a short data window centered at time t, as shown in Fig. 9.

![Windowed Fourier Transform](image)

**Fig. 9** Calculation of the Short Time Fourier Transform (upper view) and Instantaneous Spectral Density (lower view) of Vibration Signal

Spectral coefficients are computed for this short length of data. The window is then moved to a new position and that calculation repeated. If the width of this window is represented by time duration T, its frequency bandwidth is approximately 1/T.
For all calculations with time dependent spectra, the fundamental problem is that one can’t obtain a high degree of resolution in both the time and frequency domain simultaneously. This is known as uncertainty principle. A feature of the STFT is that all spectral estimates have the same (constant) bandwidth. To obtain adequate discrimination at low frequencies may involve excessively tight discrimination at higher frequencies and therefore unnecessarily long computation times. Also, if adjacent data windows overlap, the STFT generates redundant information in the sense that only a portion of the output data would be sufficient to allow the input data to be exactly regained by using the inverse STFT.

Another method is the Wigner-Ville method. The Wigner-Ville method was named after E. P. Wigner, who used it in quantum mechanics (1932), and J. Ville (1948), who used it for harmonic analysis. It is based on the following idea. Suppose that it is possible to compute an instantaneous correlation function

\[ R_\tau (\tau, t) = E[x(t - \frac{\tau}{2})x(t + \frac{\tau}{2})] \]  

(3.2.1)

at time \( t \). The operator \( E \) denotes the average of a statistical ensemble, or collection of sample functions, which we assume is available for analysis. Calculating the Fourier transform of this correlation function then gives

\[ S_\omega (\omega, t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} R_\tau (\tau, t) e^{-i\omega \tau} d\tau \]  

(3.2.2)

where the instantaneous spectral density function \( S_\omega (\omega, t) \) is a function of time \( t \) and frequency \( \omega \). In practice, it is never possible to compute an ensemble average. If that
averaging operation is omitted and equation (3.2.1) substituted in (3.2.2), a single
sample function \( x(t) \)

\[
\phi_x(\omega,t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} x(t - \frac{\tau}{2})x(t + \frac{\tau}{2})e^{-j\omega \tau} d\tau \tag{3.2.3}
\]

where \( \Phi_x(\omega,t) \) is a new function of \( t \) and \( \omega \). It is called the Wigner distribution of
\( x(t) \). The disadvantage of this method is that, although the integral in (3.2.3) is
centered at time \( t \), it covers an infinite range of \( \tau \). So, it depends on the character of \( x \)
far away from the local time \( t \). Hence, it doesn’t describe the truly local behavior of \( x \)
at time \( t \). In addition, it is impossible to have a local spectral density due to the
continuing nature of harmonic waves. This is a fundamental uncertainty principle for
time dependent spectra. High resolution can’t be obtained simultaneously in time and
in frequency.
3.3 Wavelet Transforms VS. Fourier transform

The Fast Fourier Transform (FFT) and the Discrete Wavelet Transform (DWT) are both linear operations that generate a data structure that contains $\log_2 n$ segments of various lengths, usually filling and transforming it into a different data vector of length $2^n$.

The mathematical properties of the matrices involved in the transforms are similar as well. The inverse transform matrix for both the FFT and the DWT is the transpose of the original data. As a result, both transforms can be viewed as a rotation in function space to a different domain. For the FFT, this new domain contains basis functions that are sines and cosines. For the wavelet transform, this new domain contains more complicated basis functions called wavelets, mother wavelets, or analyzing wavelets.

Both transforms have another similarity. The basis functions are localized in frequency (see Fig. 10) making mathematical tools, such as power spectra (how much power is contained in a frequency interval), useful at picking out frequencies.

![FFT Magnitude](image_url)

Fig. 10 FFT for Case 12
and calculating power distributions.

If the system is stationary and doesn’t change with time, spectral analysis that provide averaged spectral coefficient will be best to be used. However not every process is stationary and doesn’t change with time. Many random processes are essentially non-stationary, and the spectral density won’t give us enough information to identify when “in time” faults were happening. Spectral analysis can be calculated in the usual way but the results provide data about the frequency composition averaged over the duration of the signal. That is where the dissimilarity comes between FFT and wavelets.

The most interesting dissimilarity between these two kinds of transforms is that individual wavelet functions are localized in space. Fourier sine and cosine functions are not. This localization feature, along with wavelets' localization of frequency makes many functions and operators "sparse" when transformed into the wavelet domain. Hence, one of the major advantages afforded by wavelets is the ability to perform local analysis, that is, to analyze a localized area of a larger signal. For example, consider a sinusoidal signal with a small discontinuity, one so tiny as to be barely visible. A power fluctuation or a noisy switch easily could generate such a signal in the real world.

![Sinusoid with a small discontinuity](image)

**Fig. 11** Sinusoid with a Small Discontinuity
A plot of the Fourier coefficients of this signal shows nothing particularly interesting, "a flat spectrum with two peaks representing a single frequency".

However, a plot of Daubechies wavelet coefficients clearly shows the exact location in time of the discontinuity.

![Wavelet Coefficients](image)

**Fig. 12** Comparison Between Wavelet and Fourier Transform for Discontinuity Detection

Hence, Wavelet analysis is capable of revealing aspects of data that other signal analysis techniques miss: aspects like trends, breakdown points, and discontinuities in higher derivatives, and self-similarity. Furthermore, because it affords a different view of data than those presented by traditional techniques, wavelet analysis can often compress or de-noise a signal without appreciable degradation.

One way to see the time-frequency resolution differences between the Fourier transform and the wavelet transform is to look at the basis function coverage of the time-frequency plane. Fig. 13 shows a windowed Fourier transform (WFT), where the window is simply a square wave.
The square wave window truncates the sinusoid function to fit a window of a particular width. Because a single window is used for all frequencies in the WFT, the resolution of the analysis is the same at all locations in the time-frequency plane.

An advantage of wavelet transforms is that the windows vary. In order to isolate signal discontinuities, one would like to have some very short basis functions. At the same time, in order to obtain detailed frequency analysis, one would like to have some very long basis functions. A way to achieve this is to have short high-frequency basis functions and long low-frequency ones. This happy medium is exactly what you get with wavelet transforms. Fig. 14 shows the coverage in the time-frequency plane with one wavelet function, the Daubechies wavelet. The graphics shows how we can use wide windows for low frequencies, and narrow windows for high frequencies, and it will allow us to chop up our signal into appropriately scaled and shifted wavelet as shown later in Fig. 15.
Note that wavelet transforms do not have a single set of basis functions like the Fourier transform, which utilizes just the sinusoid function. Instead, wavelet transforms have an infinite set of possible basis functions. Thus wavelet analysis provides immediate access to information that can be obscured by other time-frequency methods such as Fourier analysis.

3.4 Denoising using wavelet

Denoising is a problem of recovering a true signal from incomplete, indirect or noisy data. Wavelets can help to solve this problem, through a technique called wavelet shrinkage and thresholding methods [32].

The fundamental idea behind wavelets is to analyze the signal using different scales. A basis function (The particular set of functions that are orthogonal) varies in scale by chopping up the same function or data space using different scale sizes, (see Fig. 15).
Indeed, some researchers in the wavelet field feel that, by using wavelets, one is adopting a whole new mindset or perspective in signal processing [24].

3.4.1 Definition of wavelet

Wavelets are functions that satisfy certain mathematical requirements and are used in representing data or other functions. The wavelet analysis procedure is to adopt a wavelet prototype function, called an "analyzing wavelet" or "mother wavelet". \( \Phi(x) \) defines an orthogonal basis, which is the wavelet basis used in this application:

\[
\Phi_{(s,L)}(x) = 2^{-s} \Phi(2^{-s} x - L)
\]  

(3.4.1)

The variables \( S \) and \( L \) are integers that scale and dilate the mother function to generate wavelets, such as the Daubechies wavelet family. The scale index \( S \) indicates the wavelet's width, and the location index \( L \) gives its position. Notice that the mother functions are rescaled, or "dilated" by powers of two, and translated by integers. What makes wavelet bases especially interesting is the self-similarity caused by the scales
and dilations. Once we know about the mother functions, we know everything about the basis.

To span our data domain at different resolutions, an analyzing wavelet is used as shown in equation (3.4.2):

\[
W(x) = \sum_{K=0}^{2^n-1} (-1)^K C_{K+1} \Phi_{(x,K)} (2x + K).
\] (3.4.2)

where \(W(x)\) is the scaling function for the mother function \(\Phi(x)\), and \(C_K\) are the wavelet coefficients. The wavelet coefficients must satisfy linear and quadratic constraints of the form:

\[
\sum_{K=0}^{2^n-1} C_K = 2
\]

\[
\sum_{K=0}^{2^n-1} C_K C_{K+2L} = 2 \delta_{L,0}
\]

where \(\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}\)

and \(L\) is the location index.

3.4.2 Selection of wavelet parameters

One of the most useful features of wavelets is the ease with which an engineer can choose the defining coefficients for a given wavelet system to be adapted for a
given problem. In Daubechies' original paper [33], she developed specific families of wavelet systems that were very good for representing a polynomial behavior.

It is helpful to think of the coefficients \( \{C_0, C_1, \ldots, C_k\} \) as a filter. The coefficients are placed in a transformation matrix, which is applied to a raw data vector. The coefficients are ordered using two dominant patterns, one that works as a smoothing filter (like a moving average), and one pattern that works to bring out the data's "detailed" information. These two orderings of the coefficients are called a quadrature mirror filter pair in signal processing parlance.

The transformation matrix is applied in a hierarchical algorithm, sometimes called a pyramidal algorithm. The wavelet coefficients are arranged so that odd rows contain an ordering of wavelet coefficients that act as the smoothing filter, and the even rows contain an ordering of wavelet coefficient with different signs that act to bring out the data's detail. The matrix is first applied to the original, full-length vector. Then the vector is smoothed and decimated by half and the matrix is applied again. Then the smoothed, halved vector is smoothed, and halved again, and the matrix is applied once more. This process continues until a trivial number of "smooth-smooth-smooth..." data remain. That is, each matrix application brings out a higher resolution of the data while at the same time smoothing the remaining data. The output of the discrete wavelet transform "DWT" consists of the remaining "smooth" components, and all of the accumulated "detailed" components.

Temporal analysis is performed with a contracted, high frequency version of the prototype wavelet, while frequency analysis is performed with a dilated, low frequency version of the same wavelet. Because the original signal or function can be
represented in terms of a wavelet expansion (using coefficients in a linear combination of the wavelet functions), data operations can be performed using just the corresponding wavelet coefficients. So, by choosing the best wavelets adapted to the data, or truncate the coefficients below a threshold, the data is sparsely represented. This sparse coding makes wavelets an excellent tool in the field of data compression.

3.4.3 Denoising via threshold selection

For denoising, a data set is first decomposed using wavelets. Some of the resulting wavelet coefficients correspond to details in the data set. If the details are small, they might be omitted without substantially affecting the main features of the data set. The idea of thresholding, then, is to set to zero all coefficients that are less than a particular threshold. The remaining nonzero coefficients are then used in an inverse wavelet transformation to reconstruct the data set. The technique is a significant step forward in handling noisy data because the denoising is carried out without smoothing out the sharp edges. The result is a cleaned-up signal that still shows important details.

To denoise the signal, the following steps should be applied [34]:

1) Transform the signal to the wavelet domain using Daubechies/ Symlets Wavelet.
2) Apply a threshold to the Wavelet coefficient.
3) Inverse-transform the image to the signal domain.

Calculating wavelet coefficients at every possible scale is a fair amount of work, and it generates an awful lot of data. It turns out, rather remarkably, that if we
choose scales and positions based on powers of two, we can reduce our calculation by half [35]. For many signals, the low-frequency content is the most important part. It is what gives the signal its identity. The high-frequency content, on the other hand, imparts flavor or nuance. It is for this reason that, in wavelet analysis, we often speak of approximations and details. The approximations are the high-scale, low-frequency components of the signal. The details are the low-scale, high-frequency components. The filtering process, at its most basic level, is shown in Fig. 16.

![Wavelet Filtering Process](image)

Fig. 16 Wavelet Filtering Process

Unfortunately, if we actually perform this operation on a real digital signal, we wind up with twice as much data as we started with. Suppose, for instance, that the original signal $S$ consists of 1000 samples of data. Then the approximation and the detail will each have 1000 samples, for a total of 2000. To correct this problem, we introduce the notion of downsampling [35]. This simply means throwing away every second data point, as shown on the first step in Fig. 17. The decomposition process can be iterated, so that one signal is broken down into many lower-resolution components. This is called the wavelet decomposition tree, see Fig. 17 step 3, on the left-hand side.
The second step in Fig. 17 is to remove most of the noise from the signal without losing any information. This denoising approach is called thresholding. It involves discarding only the portion of the details that is less than a certain limit. This would have the effect of cutting back the noise while leaving the details unaffected through most of their duration.

The third step is how those components can be assembled back into the original signal with no loss of information. This process is called reconstruction, or synthesis. The mathematical manipulation for synthesis is called the inverse discrete wavelet transform (IDWT). To synthesize a signal, we reconstruct it from the wavelet coefficients. Where wavelet analysis involves filtering and downsampling, the wavelet reconstruction process consists of upsampling and filtering. Upsampling is the process of lengthening a signal component by inserting zeros between samples. The low- and highpass decomposition filters (L and H), together with their associated reconstruction filters (L' and H'), form a system of what are called quadrature mirror filters.

In conclusion, this process involves three aspects: breaking up a signal to obtain the wavelet coefficients, modifying the wavelet coefficients (de-noising), and reassembling the signal from the coefficients.
1) Transform the Signal to the Wavelet Domain using Daubechies/Symlets

1000 Data Points

-500 DWT Coefficients

Detail Level 1

Setting a threshold

Detail Level 2

Detail Level 3

Fig. 17 Denoising Approach Using Wavelets
The following are some of the results, using two different methods, Daubechies and Symlets, with a SNR = 20 dB. In Fig. 18, the original signal and its corrupted version are shown first. Then the noisy data is denoised using two different Wavelets families. The lower two plots show excellent results for both methods.

Fig. 18 State Trajectory for one of the Fault Cases (case 12). Showing Two Different Methods for Denoising, Using Wavelet Transform.
Again using Daubechies and Symlets, Fig. 19 shows a second example with a different parameters, and at different SNR = 13.98 dB. The original signal and its corrupted version are shown first. Then the noisy data is denoised as shown in the lower two plots.

Fig. 19 State Trajectory for Another Fault Case (case 11). Showing Two Different Methods for Denoising, Using Wavelet Transform.
The quality of the denoising, using Wavelet transform, can be found by using the mean-square error between the original signal data and the denoised signal:

\[ e = \frac{1}{M} \sum_{i=1}^{M} \sqrt{(x - \tilde{x})^2 + (y - \tilde{y})^2} \]

where \( M \) is the number of samples of the source signal. By looking at both figures, Fig. 18 and Fig. 19, and choosing the Symlets transform, one can calculate the mean square error of the approximated signal with respect to the original, the result came out as follows:

Case 12: \[ e_{\text{case12}} = \frac{\sqrt{(5.3260 \times 10^{-5} + 5.4934 \times 10^{-5})}}{150 \pi} = 22.073 \times 10^{-6} \]

Case 11: \[ e_{\text{case11}} = \frac{\sqrt{(3.4210 \times 10^{-4} + 2.5924 \times 10^{-4})}}{150 \pi} = 52.037 \times 10^{-6} \]

As shown above, the error was very small, which indicates that the denoising was very successful. The user usually has to choose which Wavelet families will suit his application, and which one will give the best results. In our case above, case 12 had good results in both methods, but in case 11, the Symlets wavelet had better results over Daubechies Wavelet.

Noise is a result of different factors such as sensor noise, rubbing noise, and environmental noise, and can have big impact on the original signal. Data might get lost due to filtering as shown in Fig. 19. Both Figures showed successful recovery results, however Fig. 18 showed better recovery due to the signal to the noise ratio, which was 20dB.
Incipient failure detection can be achieved after denoising the signal. The next chapter will discuss the different methods on how we can detect a fault in a noisy environment.
CHAPTER IV
A MODEL-BASED APPROACH TO VIBRATION ANALYSIS AND DIAGNOSIS

4.1 Introduction

In this work we propose a statistical approach, which incorporates a dynamical model driven by the vibration data, for the purpose of monitoring and identifying fault in the rotating machinery. The basic idea is that each fault to be identified in the rotating machinery is associated with a certain set of parameters in the model. Different sets of parameters in the model will lead to different dynamics, as in the case when a fault occurs in the system causing the structure and/or parameters to change.

There are two different approaches for fault detection and monitoring using a model based approach. One of those schemes was the “state estimator” by Horattas, Adam, Loparo, and Abdul-Magied [29], which consisted of stochastic nonlinear observers, where it has been used as fault filters. Method proposed here is a novel off line and on-line model-based monitoring and diagnosis algorithm. It is based on computationally efficient algorithms for signal processing and parameter identification, initial results have been reported in [4].
The nonlinear state estimation problem [29] can be interpreted as estimating the state $z$ of a nonlinear dynamical system described by a system of first order differential equation:

$$
\ddot{z} = f(\ddot{z},t)
$$
$$
\ddot{s} = h(\ddot{z},t)
$$

given the measurement $\ddot{s}$. In the case of rotor dynamics $\ddot{z} = (\ddot{x},\ddot{y})'$ denotes the scaled rotor coordinates, while $\ddot{s}$ refers to the sensor measurements (unless otherwise stated the sensor data is assumed to be linearly related to the coordinates, i.e., $h(\ddot{z},t)$ is a linear function of $\ddot{z}$).

Luenberger [23] and [24] gave the solution for the class of linear deterministic systems. While Kalman in [34] gave the solution for the stochastic linear system.

Many researchers have worked on the development of state estimators for nonlinear and/or uncertain systems, see survey in [36] and [37]. Some methods such as global linearization were introduced, for example, in Bestle and Zeitz [38], Krener and Respondek [39], and Xia and Gao [40]. The basic idea of global linearization is to find a state transformation that makes the nonlinearity of the transformed system completely dependent upon the outputs, and then an observer is derived with linear error dynamics. Unfortunately, this method requires perfect knowledge of the system dynamics, including the structure and parameters.

State estimation has been proposed by Abdel-Magied [1] as an approach to fault detection and diagnosis of rotor dynamics. Based on the work Abu-Mahfouz [20], Abdel-Magied and Loparo [1,2] developed a stochastic nonlinear model with
additive plant and measurement noises. The rotor dynamic model exhibits both a linear behavior (viscoelastic behavior of the rotor and bearings) and a nonlinear behavior (rub impact behavior). It is well-known that a linear or extended Kalman filter (EKF) [34] can provide an unbiased estimate of the rotor state (i.e. the position of the rotor's geometric center) in the linear region of the state space, since it is an optimal least squares estimator. As the system moves into the nonlinear region, the state residual errors can be significant, particularly in chaotic attractor basin. Therefore, a nonlinear observer was constructed by Horattas, Adams, and Abdel-Magied [1, 2, 29], in an attempt to estimate the state in the nonlinear region.
4.2 Problem Formulation

Consider the following nonlinear dynamical model of a rotating machine:

\[
\frac{d\tilde{\bar{z}}}{dt} = f(\tilde{\bar{z}}, \tilde{\bar{u}}, t) \tilde{\bar{P}} + \tilde{\bar{n}}(t); \quad \tilde{\bar{z}}(0) = \tilde{\bar{z}}_0
\]

\[
\tilde{\bar{s}}(t) = h(\tilde{\bar{z}}, t)
\]

where

- \( n_0 \): Degree of freedom (e.g., 2 * number of discs)
- \( m_0 \): Number of external forces
- \( p_0 \): Number of parameters
- \( q \): Number of fault classes

and:

- \( \tilde{\bar{z}}_{m \times 1} = (z_1, z_2, ..., z_m)' \)
- \( \tilde{\bar{u}}_{m \times 1} = (u_1, u_2, ..., u_m)' \)
- \( \tilde{\bar{P}}_{p \times 1} \in P_1 \cup P_2 \cup ... \cup P_q \)
- \( P_i \cap P_j \neq 0 \) [inseparable faults]

The model considered in this work is a two-degree of freedom (2-DOF) \((n_0 = 2)\),

where the state vector \( \tilde{\bar{z}} = (x, y) \) represents the normalized position of the rotor within

the rotor-stator clearance circle. As described earlier, the clearance circle is a unit

circle at \((0,0)\), and \( m_0 \) is the number of the linear and the nonlinear external forces,

see sec. 2.3 for details. Two sets of parameters, \( P_i \) and \( P_j \), can overlap and share few

of their variables. In the simulation, four sets of parameters were studied \((q = 4)\), and

each set have five different parameters \((p_0 = 5)\). A set of fault categories,

\( FC = \{ fc_1, fc_2, ..., fc_q \} \), represents various perturbation patterns in the model

parameter values. The set of parameters \( P \) in the case of 2-DOF model
is \( P = \{\zeta, K_\theta, \mu, E, G\} \). Therefore, for each fault category \( fc_i \), there is a corresponding \( P_i \in \mathbb{R}^5 \) in the parameter space. This particular \( P_i \) is associated with the dynamic change of the system caused by the fault, \( fc_i \). Due to the noise, disturbances and modeling errors, the fault domains, \( P_i, i = 1, \ldots, q \), are not necessarily disjoint or separable, i.e., \( P_i \cap P_j \neq 0 \).

Let \( fc^* \) be the unknown fault category of an actual machine with the corresponding vector of parameter values \( p^* = (\zeta^*, K_\theta^*, \mu^*, E^*, G^*) \in P^* \). Clearly, due to the noise, disturbances and modeling errors, the vector \( p^* \) is a random entity.

The objective of the fault monitoring and identification algorithm is to identify the fault condition \( fc \), that best matches \( fc^* \). One possible approach is the one-match criterion [42], which is based on the similarity between the model and collected vibration signals in time domain. Given the actual vibration time-series:

\[
V^*(k) = \begin{bmatrix} s^*(k\Delta t) & u^*(k\Delta t) \end{bmatrix}; \quad k = 0, 1, 2, \ldots, T.
\]

find an index \( i \in \{1, 2, \ldots, p\} \) such that:

\[
\text{Min } \|V^* - V_i\|
\]

where \( V_i, i = 1, 2, \ldots, p \) is the data generated from the models of the machine under different fault categories. The problem with this approach is that it is sensitive to noises and disturbances, which are quite significant in our application.

Based on the above consideration, we proposed the following model-based criteria fault monitoring and identification: Given the input-output data of the process,
perform the parameter identification to obtain \( p^* = (\zeta^*, K_p^*, \mu^*, E^*, G^*) \in P^* \), then find the \( P_i \) that satisfies

\[
\text{Min} \| P^* - P_i \|
\]

Note that to continuously monitor the onset and progression of a fault, the resulting algorithm should be a recursive one.

Focusing our attention on the dynamics of rub impact defined in the previous section, two types of faults are considered:

1. External faults, such as a change in the unbalanced, or a change in the static load, which might be applied to the system.

2. Bearing faults, such as a change in either the damping \( \xi \) or the stiffness \( K_s \).

In view of this formulation, faults are associated with changes in the system parameters. The class of known faults can be defined and related to an equivalent set of parameters. A novel approach will be implemented to track these faults. It will be implemented as off-line and on-line algorithms. One should keep in mind that the system is linear in the parameters but not in the state. Some of the problems to be overcome are:

- Faults can induce nonlinear dynamics in the bearing response, which complicates the detection problem because of the sensitivity of the response to the initial conditions and/or parameters.

- Only partial measurements (displacements) of the bearing motion are to be measured, since the linear velocities are not often available.
• Measurements are contaminated with noise with potentially low signal to noise ratio (SNR). Moreover, the noise is often correlated with the measurements.

• Upon faults occurrence, induced nonlinearity due to rub impact produce high frequency harmonics.

• Uncertainty in the model parameters, process disturbances, and the effect of unmodeled dynamics complicate the detection/diagnosis problem.

  In general, the desired functions of the diagnostic system are:

• Detection of the fault.

• Diagnosis or isolation of the fault.

• Identification of the size (level) and the time of the fault.

  The design requirements are:

• Small detection delay time, which is minimized for fixed false alarm rate.

• High rate of correct detection.

• Low rate of false alarm.

• Isolatability: The ability to distinguish (isolate) faults.

• Sensitivity: The identification of the size (level) of the faults under different conditions of SNR.

• Robustness: The ability to isolate faults in the presence of modeling errors and/or unknown disturbance.

The objective is to include all these design factors in the development of the proposed diagnostic system. Some of the design factors will be further discussed in the simulation section.
4.3 Proposed Approach

The following diagram shows the different approaches to fault detection and diagnosis:

![Diagram](image)

Fig. 20 Proposed Approach

4.3.1 Off-line Parametric identification

The offline parameter identification algorithm is based on the gradient descent approach. The following is a description of the gradient descent approach and its use in the identification algorithm.
Let "P" be the parameter to be estimated, and J be the mean square error "MSE" of the two degrees of freedom (2 DOF) in the rotor position, (there are many other variation on mean square error derivation, i.e., adding what is called *forgetting factor*), then one can derive the necessary condition of optimality of the parameter vector "P" as follows:

\[ J = \| e \|_2 \]

\[
J = \sqrt{\frac{1}{2T} \sum_{t_0}^{t_r-\tau} (x - \hat{x})^2 + (y - \hat{y})^2}
\]

\[
\frac{\partial J}{\partial P} = 0 \Rightarrow \frac{1}{2T} \sum_{t_0}^{t_r-\tau} \frac{\partial}{\partial P} (x - \hat{x})^2 + \frac{\partial}{\partial P} (y - \hat{y})^2 = 0
\]

\[
\frac{1}{2T} \sum_{t_0}^{t_r-\tau} \frac{\partial}{\partial x} (x - \hat{x})^2 \frac{\partial x}{\partial P} + \frac{\partial}{\partial y} (y - \hat{y})^2 \frac{\partial y}{\partial P} = 0
\]

\[
\frac{1}{T} \sum_{t_0}^{t_r-\tau} (x - \hat{x}) \frac{\partial x}{\partial P} + (y - \hat{y}) \frac{\partial y}{\partial P} = 0
\]

\[
(x - \hat{x}) \frac{\partial x}{\partial P} = 0 \quad \text{and} \quad (y - \hat{y}) \frac{\partial y}{\partial P} = 0
\]

Adopting the steepest descent algorithm to optimize the parameter "P", one would get the following differential equation, with \( A_x \) and \( A_y \) (\( B_x \) and \( B_y \) in the discrete-time version) being inversely related to the sampling rate.

\[
\dot{P} = A_x (x - \hat{x}) \frac{\partial x}{\partial P} + A_y (y - \hat{y}) \frac{\partial y}{\partial P}
\]

The parameter vector "P" will converge whenever the error "e" is minimized.
The above equation can be digitized as:

\[ P(k+1) = P(k) - B_x(k) \frac{\partial x}{\partial P}(k)(-x(k) + \hat{x}(k)) - B_y(k) \frac{\partial y}{\partial P}(k)(-y(k) + \hat{y}(k)) \]  \hspace{1cm} (4.3.1.1)  

where:

\( P \): The parameter to be estimated, i.e., \( \zeta, K_\beta, \mu, E, \omega, G^* \).

\( \frac{\partial x}{\partial P} \) and \( \frac{\partial y}{\partial P} \): The gradients of \( (x, y) \) with respect to \( P \), as calculated below.

\( (x, y) \): Data from the process.

\( (\hat{x}, \hat{y}) \): Estimates from the model.

\( B_x(k) \) and \( B_y(k) \): Adaptation gains (usually monotonically increasing).

\( k \): Adaptation step.

The algorithm is said to converge when \( \| P(k+1) - P(k) \| \leq \varepsilon \) or

\( \|(\hat{x}, \hat{y})(k+1) - (\hat{x}, \hat{y})(k)\| \leq \varepsilon \), where \( \varepsilon \) is a predetermined small number.

The objective here is to calculate gradients of the equation of motion with respect to parameters. Once obtained, the gradients can be used in the parameter estimator to estimate the parameter values from the data.

Recall that the mathematical model for the rigid impact dynamics is:

\[
\begin{align*}
\ddot{x} + 2\zeta \dot{x} + x &= -\left(K_p \delta^a (\cos \theta - \mu \sin \theta)\right) + E\omega^2 \cos \omega t; \\
\ddot{y} + 2\zeta \dot{y} + y &= -\left(K_p \delta^a (\sin \theta + \mu \cos \theta)\right) + E\omega^2 \sin \omega t - G^*; \\
\end{align*}
\]

where
\[ x(t_0) = x_0; \quad y(t_0) = y_0 \]

\[ \theta = \tan^{-1}(y/x), \quad \delta = \frac{\sqrt{x^2 + y^2}}{C} - 1 \]

*therefore*, \( \cos \theta = \frac{x}{\sqrt{x^2 + y^2}}, \quad \sin \theta = \frac{y}{\sqrt{x^2 + y^2}} \)

and "x and y" denote the points of the shaft.

Both equation (2.3.2) and (2.3.3) can be rewritten in state space form, using

the following choice of state variables: \( x_1 = x, \quad x_2 = \dot{x}, \quad x_3 = y, \quad x_4 = \dot{y} \)

then

\[
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -x_1 - 2\xi x_2 + E\omega^2 \cos \omega t + F_x \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= -x_3 - 2\xi x_4 + E\omega^2 \sin \omega t + F_y - G^* 
\end{aligned}
\]

(4.3.1.2)

where

\[
F_x = -K_\beta \delta^a (\cos \theta - \mu \sin \theta) \\
F_y = -K_\beta \delta^a (\sin \theta + \mu \cos \theta)
\]

The sets of equation (4.3.1.2) were used to solve both equation (2.3.2) and (2.3.3) in the Matlab\textsuperscript{®} implementation of (4.3.1.1).

Also, the following two formulae will be used in all equations of motion of the gradient variable:
\[
\begin{align*}
\frac{\partial}{\partial x} (\cos \theta - \mu \sin \theta) &= (-\sin \theta - \mu \cos \theta) \frac{\partial \theta}{\partial x} \\
\frac{\partial}{\partial x} (\frac{\sin \theta}{\cos \theta}) &= \frac{\partial}{\partial x} \left( \frac{y}{x} \right) \\
\frac{\partial}{\partial \theta} \left( \frac{\sin \theta}{\cos \theta} \right) \frac{\partial \theta}{\partial x} &= -\frac{y}{x^2} \\
\frac{1}{\cos^2 \theta} \frac{\partial \theta}{\partial x} &= -\frac{y}{x^2} \\
\frac{\partial \theta}{\partial x} &= -\frac{y}{x^2 \cos^2 \theta} \\
\text{therefore } \frac{\partial}{\partial x} (\cos \theta - \mu \sin \theta) &= (\sin \theta + \mu \cos \theta) \frac{y}{x^2 \cos^2 \theta} \\
\frac{\partial \delta^a}{\partial x} &= \alpha \frac{1}{C} \frac{x}{\sqrt{x^2 + y^2}} \delta^{a-1} \\
&= \frac{\alpha}{C} \cos \theta \delta^{a-1}
\end{align*}
\]

A similar equation can be obtained for the corresponding derivative terms with respect to \(y\).

\[
\begin{align*}
\frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) &= (\cos \theta - \mu \sin \theta) \frac{\partial \theta}{\partial y} \\
\frac{\partial}{\partial y} (\frac{\sin \theta}{\cos \theta}) &= \frac{\partial}{\partial y} \left( \frac{y}{x} \right) \\
\frac{\partial}{\partial \theta} \left( \frac{\sin \theta}{\cos \theta} \right) \frac{\partial \theta}{\partial y} &= \frac{1}{x} \\
\frac{1}{\cos^2 \theta} \frac{\partial \theta}{\partial y} &= \frac{1}{x} \\
\frac{\partial \theta}{\partial y} &= \frac{1}{x} \cos^2 \theta \\
\text{therefore } \frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) &= (\cos \theta - \mu \sin \theta) \frac{1}{x} \cos^2 \theta
\end{align*}
\]
\[ \frac{\partial \delta^\alpha}{\partial y} = \alpha \frac{1}{C} \frac{y}{\sqrt{x^2 + y^2}} \delta^{\alpha-1} = \frac{\alpha}{C} \sin \theta \delta^{\alpha-1} \]

To calculate the gradient trajectory one obtains differential equations for

\[ \frac{\partial x(t)}{\partial P} \quad \text{and} \quad \frac{\partial y(t)}{\partial P} \] (where P represents one of the parameters), and solve their along

with the system's equation of motion to minimize mean squared error between data

and the model. The derivation of the gradient of \( x \) and \( y \) with respect to the

parameters are listed:

a) To obtain the ODE for \( \frac{\partial x(t)}{\partial \zeta} \)

\[ \theta = \tan^{-1}(y/x), \quad \delta = \frac{\sqrt{x^2 + y^2}}{C} - 1 \]

\textbf{therefore} \( \cos \theta = \frac{x}{\sqrt{x^2 + y^2}} \quad \text{sin} \theta = \frac{y}{\sqrt{x^2 + y^2}} \)

\[ \frac{\partial x}{\partial \zeta} + 2 \zeta \frac{\partial x}{\partial \zeta} + \frac{\partial x}{\partial \zeta} = 2 \dot{x} - K_{\beta} \left\{ \left( \frac{\partial \delta^\alpha}{\partial x} (\cos \theta - \mu \sin \theta) + \delta^\alpha \frac{\partial}{\partial x} (\cos \theta - \mu \sin \theta) \right) \frac{\partial x}{\partial \zeta} \right\} ; \]

\[ \frac{\partial x}{\partial \zeta}(t_0) = 0 \]

b) To obtain the ODE for \( \frac{\partial y(t)}{\partial \zeta} \)
\[ \frac{\delta y}{\delta \zeta} + 2 \zeta \frac{\delta y}{\delta \zeta} + \frac{\delta v}{\delta \zeta} = -2 \dot{y} - K_p^* \left( \frac{\partial \delta^*}{\partial x} (\sin \theta + \mu \cos \theta) + \delta^* \frac{\partial}{\partial x} (\sin \theta + \mu \cos \theta) \right) \frac{\partial x}{\partial \zeta} + \right. \]

\[ \left. \left( \frac{\partial \delta^*}{\partial y} (\sin \theta + \mu \cos \theta) + \delta^* \frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) \right) \frac{\partial y}{\partial \zeta} \right) ; \]

\[ \frac{\partial y}{\partial \zeta} (t_0) = 0 \]

c) To obtain the ODE for \( \frac{\partial x(t)}{\partial K_p^*} \)

\[ \frac{\partial x}{\partial K_p^*} + 2 \zeta \frac{\partial x}{\partial K_p^*} + \frac{\partial x}{\partial K_p^*} = \left[ \delta^* (\cos \theta - \mu \sin \theta) \right. \]

\[ \left. + K_p^* \left( \frac{\partial \delta^*}{\partial x} (\cos \theta - \mu \sin \theta) + \delta^* \frac{\partial}{\partial x} (\cos \theta - \mu \sin \theta) \right) \frac{\partial x}{\partial K_p^*} + \right. \]

\[ \left. \left( + K_p^* \left( \frac{\partial \delta^*}{\partial y} (\cos \theta - \mu \sin \theta) + \delta^* \frac{\partial}{\partial y} (\cos \theta - \mu \sin \theta) \right) \frac{\partial y}{\partial K_p^*} \right) ; \]

\[ \frac{\partial x}{\partial K_p^*} (t_0) = 0 \]

d) To obtain the ODE for \( \frac{\partial y(t)}{\partial K_p^*} \)

\[ \frac{\partial y}{\partial K_p^*} + 2 \zeta \frac{\partial y}{\partial K_p^*} + \frac{\partial y}{\partial K_p^*} = \left( \delta^* (\sin \theta + \mu \cos \theta) + \right. \]

\[ \left. \left( K_p^* \left( \frac{\partial \delta^*}{\partial y} (\sin \theta + \mu \cos \theta) + \delta^* \frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) \right) \frac{\partial y}{\partial K_p^*} \right) ; \]

\[ \frac{\partial y}{\partial K_p^*} (t_0) = 0 \]

e) To obtain the ODE for \( \frac{\partial x(t)}{\partial \mu} \)
\[
\frac{\partial x}{\partial \mu} + 2\zeta \frac{\partial x}{\partial \mu} + \frac{\partial x}{\partial \epsilon} = -K_p \delta \sin\theta \left\{ -K_p \delta \left( \cos\theta - \mu \sin\theta \right) \right\} \frac{\partial x}{\partial \mu} \\
+ K_p \left( \frac{\partial \delta}{\partial x} \left( \cos\theta - \mu \sin\theta \right) + \delta \frac{\partial}{\partial x} \left( \cos\theta - \mu \sin\theta \right) \right) \frac{\partial x}{\partial \mu} \\
+ K_p \left( \frac{\partial \delta}{\partial y} \left( \cos\theta - \mu \sin\theta \right) + \delta \frac{\partial}{\partial y} \left( \cos\theta - \mu \sin\theta \right) \right) \frac{\partial y}{\partial \mu} \\
\frac{\partial x}{\partial \mu}(t_0) = 0
\]

f) To obtain the ODE for \( \frac{\partial y(t)}{\partial \mu} \)

\[
\frac{\partial y}{\partial \mu} + 2\zeta \frac{\partial y}{\partial \mu} + \frac{\partial y}{\partial \epsilon} = -K_p \delta \cos\theta + \\
K_p \left( \frac{\partial \delta}{\partial y} \left( \sin\theta + \mu \cos\theta \right) + \delta \frac{\partial}{\partial y} \left( \sin\theta + \mu \cos\theta \right) \right) \frac{\partial y}{\partial \mu} \\
+ K_p \left( \frac{\partial \delta}{\partial x} \left( \sin\theta + \mu \cos\theta \right) + \delta \frac{\partial}{\partial x} \left( \sin\theta + \mu \cos\theta \right) \right) \frac{\partial x}{\partial \mu} \\
\frac{\partial y}{\partial \mu}(t_0) = 0
\]

g) To obtain the ODE for \( \frac{\partial x(t)}{\partial \epsilon} \)

\[
\frac{\partial x}{\partial \epsilon} + 2\zeta \frac{\partial x}{\partial \epsilon} + \frac{\partial x}{\partial \epsilon} = -K_p \delta \cos\theta \left\{ -K_p \delta \left( \cos\theta - \mu \sin\theta \right) \right\} \frac{\partial x}{\partial \epsilon} + \omega^2 \cos\theta \\
\frac{\partial x}{\partial \epsilon}(t_0) = 0
\]

h) To obtain the ODE for \( \frac{\partial y(t)}{\partial \epsilon} \)
\[
\frac{\ddot{y}}{\partial E} + 2\xi \frac{\dot{y}}{\partial E} + \dot{y} = -K_p \left( \frac{\partial \delta^x}{\partial y} (\sin \theta + \mu \cos \theta) + \delta^x \frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) \right) \frac{\dot{y}}{\partial E} + \omega^2 \sin \omega t;
\]

\[
\frac{\dot{y}}{\partial E} (t_0) = 0
\]

i) If the actual rotating speed is unknown then it can be treated as a parameter. To obtain the ODE for \( \frac{\partial x(t)}{\partial \omega} \)

\[
\frac{\ddot{x}}{\partial \omega} + 2\xi \frac{\dot{x}}{\partial \omega} + \dot{x} = -K_p \left( \frac{\partial \delta^x}{\partial x} (\cos \theta - \mu \sin \theta) + \delta^x \frac{\partial}{\partial x} (\cos \theta - \mu \sin \theta) \right) \frac{\dot{x}}{\partial \omega} + \omega \omega (2 \omega \cos \omega t - \omega^2 \tau \sin \omega t);
\]

\[
\frac{\dot{x}}{\partial \omega} (t_0) = 0
\]

j) To obtain the ODE for \( \frac{\partial y(t)}{\partial \omega} \)

\[
\frac{\ddot{y}}{\partial \omega} + 2\xi \frac{\dot{y}}{\partial \omega} + \dot{y} = -K_p \left( \frac{\partial \delta^y}{\partial y} (\sin \theta + \mu \cos \theta) + \delta^y \frac{\partial}{\partial y} (\sin \theta + \mu \cos \theta) \right) \frac{\dot{y}}{\partial \omega} + \omega \omega (2 \omega \sin \omega t + \omega^2 \tau \cos \omega t);
\]

\[
\frac{\dot{y}}{\partial \omega} (t_0) = 0
\]

k) To obtain the ODE for \( \frac{\partial x(t)}{\partial G^*} \)
\[
\frac{\partial \delta}{\partial \xi} \frac{\partial \delta}{\partial \eta} + 2 \zeta \frac{\partial \delta}{\partial \xi} + \frac{\partial \delta}{\partial \eta} = -K_p \left( \frac{\partial \delta}{\partial x} \delta (\cos \theta - \mu \sin \theta) + \delta \frac{\partial \delta}{\partial x} (\cos \theta - \mu \sin \theta) \frac{\partial x}{\partial \xi} \right) + \left( \frac{\partial \delta}{\partial y} \delta (\cos \theta - \mu \sin \theta) + \delta \frac{\partial \delta}{\partial y} (\cos \theta - \mu \sin \theta) \frac{\partial y}{\partial \xi} \right) \right); \\
\frac{\partial x}{\partial \xi} (t_0) = 0 \\
\Rightarrow \frac{\partial x}{\partial \xi} (t) = 0
\]

l) To obtain the ODE for \( \frac{\partial y(t)}{\partial \xi} \)

\[
\frac{\partial \delta}{\partial \eta} \frac{\partial \delta}{\partial \eta} + 2 \zeta \frac{\partial \delta}{\partial \eta} + \frac{\partial \delta}{\partial \eta} = -K_p \left( \frac{\partial \delta}{\partial y} \delta (\sin \theta + \mu \cos \theta) + \delta \frac{\partial \delta}{\partial y} (\sin \theta + \mu \cos \theta) \frac{\partial y}{\partial \eta} \right) \left( \frac{\partial \delta}{\partial x} \delta (\sin \theta + \mu \cos \theta) + \delta \frac{\partial \delta}{\partial x} (\sin \theta + \mu \cos \theta) \frac{\partial x}{\partial \eta} \right) \right) - \zeta; \\
\frac{\partial y}{\partial \eta} (t_0) = 0
\]

Hence, we can use the gradient equations of motion along with the system's equations of motion to guide the parameter optimizer to minimize the Mean Squared Error.

The gradient descent method is very attractive, because it is simple to implement and lends itself for adaptive parameter estimation but suffers from slow convergence.

The following block diagram is a simple representation for Matlab program, which was written to implement the algorithm.
4.3.2 On-line Parameter Estimation

The above off-line algorithm is suitable for handling general nonlinear dynamics that involve nonlinear functions of both the state variables and the parameters, i.e., \( \dot{x} = \phi(x, u, P, t) \). But it is not suitable for real-time fault monitoring, because of the computational complexity.

To develop an on-line diagnosis algorithm for rotor dynamics, one should look for simplification of the dynamic equations that could yield a simple recursive online estimation algorithm. This section describes how such an algorithm can be developed by exploiting the linearity of the mathematical model (4.3.1.2) with respect to parameters. Recall that the mathematical model for the rub impact is:

\[
\ddot{x} + 2\zeta \dot{x} + x = -\left( K_\theta^* \delta(x - \mu \sin \theta) \right) + E \omega^2 \cos \omega t;
\]

\[
\ddot{y} + 2\zeta \dot{y} + y = -\left( K_\theta^* \delta(y + \mu \cos \theta) \right) + E \omega^2 \sin \omega t - G^*;
\]
and that the set of parameters $\xi, K_\beta, \mu, E, \text{ and } G^*$ provides complete information about all modes of the rotor-stator system, including the normal and different fault modes. Therefore, if the parameters of the model can be accurately and recursively estimated from real-time vibration data, then the diagnosis task can be reduced to that of recursive parameter estimation and pattern classification.

For all practical purposes, the output map between the rotor position $[x(t), y(t)]$ and the data from the vibration sensors is linear, and hence one can assume that map to be the identity map. Then, from the above equations, the nonlinear rotor-stator model can be simplified and represented in the following linear input-output model,

$$s(t) = \psi(t)P(t-1) + n(t)$$  \hspace{1cm} (4.3.2.1)

where,

$$s(t) = [x(t), y(t)]'$$
$$\psi(t) = 
\begin{bmatrix}
-2\dot{x} & -\langle \delta^x \cos \theta \rangle & \langle \delta^x \sin \theta \rangle & \omega^2 \cos \omega \tau & 0 & -\ddot{x} \\
-2\dot{y} & -\langle \delta^y \sin \theta \rangle & \langle \delta^y \cos \theta \rangle & \omega^2 \sin \omega \tau & -1 & -\ddot{y}
\end{bmatrix}$$

$$P(t) = [\xi, K_\beta, \mu, E, G, 1]$$

$$n(t) : \text{Noise}$$

A natural predictor of the vibration data $s(t)$ is then, $\hat{s}(t) = \psi^T(t)\hat{P}(t-1)$ and a typical recursive estimation algorithm, by Oliver and Martin [41], is

$$\hat{P}(t) = \hat{P}(t-1) + K(t)(s(t) - \hat{s}(t))$$  \hspace{1cm} (4.3.2.2)
Here $\hat{P}(t)$ is the parameter estimate at time $t$, and $s(t)$ is the observed 2-D vibration data at time $t$. The estimate $\hat{s}(t)$ is a prediction of the 2-D vibration data $s(t)$ based on observations up to time $(t-1)$, and the parameter estimate at time $(t-1)$. The gain $K(t)$ is the Kalman gain, it is typically chosen as $K(t) = Q(t) \psi(t)$. Other efficient algorithms can also be applied (such as the one in system identification toolbox in Matlab) to provide unbiased estimates.

The elements of the $\psi(t)$ matrix can be obtained from the vibration data at time $t$ using state-space filters. To avoid modeling the nonstationary noise in the ARMAX model, a wavelet-based algorithm was used for denoising the vibration signals before it is fed to the parameter estimator.

Fig. 22 shows the block diagram for the on-line method. Different parameters were stored and categorized into different cases (N, F1, F2, F3...), where "N" represent the normal case (no fault), and the "Fi" for i=1 to 7, are the different fault cases. These parameters were passed to the process. The process is the two-degree of freedom mathematical model, which was introduced in chapter II (see equation 2.3.2 and 2.3.3). While the process generating the x and y data, the model with an initial parameter values will generate $\hat{x}$ and $\hat{y}$. The model has the same mathematical presentation as the process with unknown parameters. Once the model finishes generating its data, it compares the output values of the process to the output data of the model and generates the estimated values of the parameters. The system will converge when the error is minimized between the two output values. The output of the parameter estimator is used to classify the vibration data using the nearest neighbor algorithm, as shown in Fig. 22. However, in some practical applications, the
impact of the noise and compound faults on the parameter estimate can lead to a lower classification rate. In those cases, a Kolmogorov complexity-based classifier, such as the one in Abu-Mahfouz [20], can be used.

Note that the linearity assumption is valid, in general, and the on-line method should work well under this condition, as shown in Chapter V. However, if the parameters are nonlinear, then the off-line method is more suitable.

The purpose of monitoring and diagnostics system is to detect the different abnormalities and faults and isolate them when they happen. Then we can identify the level of incipient faults when they occur and use the fault information to help maintenance scheduling on an as needed basis.
Fig. 22  Block Diagram of the Proposed Model-Based Diagnosis Algorithm
CHAPTER V
IMPLEMENTATION AND SIMULATION RESULTS

5.1 Introduction

In this chapter, we discuss the implementation issues associated with the proposed fault detection scheme. Also, we present computer simulations of different fault scenarios under different parameter sets. Then, we show that the proposed algorithm has a high degree of robustness to model uncertainty in the nonlinear part of the model that captures the rubbing phenomena, as well as, low sensitivity to the effect of measurement noise. Our goal is to be able to distinguish one rotor fault case from another by defining the fault signatures of a nonlinear process using parameter estimation approach under different signal to noise conditions.

5.2 Off-line Implementation and Simulation Results

The mathematical models derived in the previous chapters are used to build the simulation for different fault scenarios. When the mathematical model in equation 2.3.2 and 2.3.3 is simulated, it goes through an initialization phase and then a repetitive simulation cycle. The initialization phase starts by setting the initial
conditions to small random numbers or to zeros, depending in which region we are trying to simulate. In our simulations, random initial conditions were used in the case of monitoring the steady state behavior of the rotor, while zeros were used in the case of transient behavior.

Next, by specifying the six parameters ($\zeta$, $K_\theta$, $\eta$, $E$, $\omega$ and $G$) of the mathematical model, and for simplicity, we specify the period to be $2\pi$ (Fig. 23), we can solve the ODE and store the values of $x$, $\dot{x}$, $y$, $\dot{y}$.

![Time History of X and Y](image)

Fig. 23 Time History of $x$ and $y$ for a Period of $2\pi$

Noises were added to the measurement data in order to make the simulator more realistic. Wavelet-based denoising was used to clean up the signal as discussed in Chapter III.

Next, explicit initialization values for the parameters are specified. Similarly initial values were assigned to the signals ($x$ and $y$), i.e., initial values of '0s' were assigned when monitoring the rotor's transient behavior.
After the initialization phase, simulation cycles are run. Each run consists of the following steps:

- Solve both ODE “equation (2.3.2) and (2.3.3)” and get an estimated \( x, \dot{x}, y, \dot{y} \).
- Get the gradient of the \( \zeta, K_\beta, \eta, E, \omega \) and \( G^* \) by using the equations in Chapter IV and applying the same subroutine as step 1.
- Calculate new parameters “\( P_{k+1} \)” values by using the \( x \) and \( y \), Estimated \( x \) and \( y \), and the old parameters “\( P_k \)” as described in section 4.3.
- Check if the new parameters \( P_{k+1} \) minus the old parameters \( P_k \) are smaller than \( \varepsilon \) for all \( k \), where \( \varepsilon \) is a very small value.
- Parameters are updated.
- Repeated simulation cycles are continued until the Parameters reach its desired values.

The following flowchart, Fig. 24, explains the above procedure step by step, see Appendix D for details code:
Fig. 24 Flowchart Diagram for Off-Line Diagnosis Algorithm
Let us choose two different parameters "\( \zeta = 0.14 \)" and "\( E = 0.49 \)" for one of the fault cases (Case 12). By following the procedure in Fig. 24, the following results were obtained, as shown in Fig. 25:

![Graphs showing the amplitude of \( \zeta \) and \( E \) over iterations.](image)

**Fig. 25** Value of \( \zeta \) and \( E \). Using Gradient Descends Approach.

Notice that the target values of \( \zeta \) and \( E \) were obtained after 1800 iterations. Both values were very close to the actual values "\( \zeta = 0.1396 \) and \( E = 0.4896 \)". Although the convergence speed is a little slow, the values of the parameters were very close to the target. In addition, we can control the convergence speed by controlling the gain, \( B_x \) and \( B_y \), in the gradient descends equation (4.3.1.1). The selection of the gain is very important, since too high of a gain will cause an oscillation or instability.
The following is the state trajectory for the same case:

Fig. 26 State Trajectory for Case 12, Where an Off-line Algorithm Was Used
A simple block diagram Fig. 27 shows the flow of the denoising:

As shown in Fig. 27, the denoising algorithm is successful in obtaining the original signal from a noisy distorted signal of Case 12. It was achieved by applying the wavelet algorithm to the output of the "Rotor_Process" which has the vector $x$, vector $y$, vector $Z$, and vector $V$ (regressors). Furthermore, the denoising algorithm was applied only to the following outputs: $x(1)= x, x(3)= y, and Z(6)= \bar{x}, V(6)= \bar{y}$. 
5.3 On-line Simulation Results

To test the proposed algorithm, simulation experiments for the model given by Abu-Mahfouz [20] have been conducted. The goal is to classify real-time vibration data into one of the fault categories and dynamic modes, shown in Table I. To demonstrate the performance of the diagnosis algorithm, the rotor-stator process was simulated under different fault conditions, such as the quasi-periodic and chaotic dynamics, different signal to noise ratios, and random initial conditions.

Table I. Fault Categories and Sample Simulation Parameter Values

<table>
<thead>
<tr>
<th>Fault category</th>
<th>Dynamic Mode</th>
<th>Sample Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swinging with Chattering. Rotor rubbing with the bottom section of the clearance.</td>
<td>Harmonics-Order 1</td>
<td>[0.20, 0.614E4, 0.13, 0.40, 1.58]</td>
</tr>
<tr>
<td></td>
<td>Harmonics-Order 2 Chaotic</td>
<td>[0.35, 1.224E4, 0.68, 0.31, 1.20]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[0.20, 1.254E4, 0.13, 0.31, 1.23]</td>
</tr>
<tr>
<td>Rub impact causing wear around N points.</td>
<td>Harmonics</td>
<td>[0.16, 1.254E4, 0.30, 0.75, 0.123]</td>
</tr>
<tr>
<td></td>
<td>Chaotic</td>
<td>[0.032, 1.004E4, 0.1, 0.80, 0.981]</td>
</tr>
<tr>
<td></td>
<td>Harmonic</td>
<td>[0.14, 1.334E4, 0.30, 0.45, 0.131]</td>
</tr>
<tr>
<td>Rub impact all around the clearance sector</td>
<td>Harmonics-Order 20</td>
<td>[0.024, 1.389E4, 0.13, 0.34, 0.136]</td>
</tr>
<tr>
<td></td>
<td>Chaotic</td>
<td>[1.354, 1.124E5, 0.13, 0.32, 1.1]</td>
</tr>
<tr>
<td></td>
<td>Quasi-Periodic</td>
<td>[0.14, 1.334E3, 0.50, 0.49, 0.13]</td>
</tr>
</tbody>
</table>

Let's choose the normal case of operation for the rotating machine, so we can compare it with the other faulty cases. The normal case has the following set of parameters:
Fig. 28 Parameters for the Normal Case

Fig. 28 was extracted from the block diagram in the Matlab Simulink.

Fig. 29 (a) and (b) show the orbit diagram for both the original and the simulated model. Random initial conditions were applied, and the simulated model was able to track and classify the parameters of the nonlinear system Fig. 30.

Fig. 29 (a) Original Signal (b) Simulated Model Signal
Subsequently, a fault is introduced to the system with the following parameters:

\[
[\omega \ E \ \mu \ K_\theta \ \xi \ K_\delta \ G] = [1.0 \ 0.49 \ 650 \ 0.14 \ 1330 \ 0.3]
\]

By applying those parameters to our system, nonlinear quasi-periodic behaviors were observed as shown in Fig. 31.
Again our simulated model was able to track the parameters changes in the system as shown in Fig. 32.

And the parameters convergence are shown in Fig. 33.
For clarity, a zoomed version of the parameters convergence is shown in Fig. 34.

Fig. 34 Parameters Convergence

Fig. 35 Parameters Convergence
Next, a noise was induced to the system. The signal applied to the system has a SNR = 20dB. In addition, the simulations assume an additive noise due to the rubbing throughout the linear region.

Fig. 36 (a) shows the original data with the noise, while Fig. 36 (b) shows the capability of the denoising algorithm to filter out the noise and follow the orbit diagram of the system.

![Fig. 36 State Trajectory (a) Additive Noise and (b) Simulated Model](image)

Again, the parameters' convergence is shown in Fig. 37.
Fig. 37 Parameters Convergence in a Noisy Environment

Fig. 38 shows a zoomed version of the parameters convergence.

Fig. 38 Parameters Estimation
In general, experimental simulation data verified on-line parameters estimation algorithm and demonstrated it can tolerate significant changes in the system. Next, abrupt changes in the values of the parameters were introduced every 100 seconds (2000 samples at the sampling period of 0.05 second). An additive sensor noise at SNR=20dB and an additive process noise (periodic and rub impact components) were applied to the model. Fig. 39 shows the Simulink block diagram for all four different cases, see Appendix D for more detailed description on both Rotor_Estimator and Rotor_Process.
Fig. 39 Block Diagram for Four Different Cases

The block diagram in Fig. 39 shows the flow of the proposed algorithm where different cases can be added. The block diagram shows only four different cases "case 1, case 5, case 12, and case 17". The cases were multiplexed, allowing one case
to be passed to the rotor process at a time. By applying a double click on each block, Simulink will open the content of that block as shown by the arrow.

"Rotor_process_c" block will take the parameters as an input, and generate all the data \( x, y, \dot{x}, \text{and} \ y \) as an input to the "Rotor_Estimator_c" block. Each block is a S_Function, which is a Matlab program, where it can be modified to serve our purpose (see Appendix D). The Rotor_Estimator_c will take the \( x \) and \( y \) data and estimate the parameters, which are then classified into different fault cases. Notice again that "Rotor_Estimator_c" has only \( x, y \) data, and the algorithm has to automatically determine which fault cases occurred.

The response of the parameter estimator is displayed in Fig. 40. The estimator

![Image](image_url)

*Fig. 40  Parameter Estimates for a Sequence of Different Fault Modes.*

was able to robustly track the parameter values in all cases as shown. The clock in
Fig. 39 can be specified by the user, this allows control over the time of the faults. The first two faults were introduced at $t = 0$, and $t = 100$ sec, respectively. To show the robustness of the algorithm at steady state, a third case were introduced at $t = 200$ sec.

Fig. 41 shows the orbit diagram for the model (without noise), the simulated data and the filtered data, during the second segment of the experiment [2000,4000]. The predictor output correlates with the original except during the estimator's transient response at the rub points with the clearance circle (0.2).
Fig. 41 State Trajectory / Orbit Diagram for Fault Mode [2000,4000].
The 2-norm is used to evaluate the error:

$$\|e\|_2 = \sqrt{(\theta - \tilde{\theta})(\theta - \tilde{\theta})^T}$$

where $(\theta - \tilde{\theta})$ is the difference between the actual and the estimated parameters respectively. Table II shows that the error of the estimation is consistently small for all four cases.

<table>
<thead>
<tr>
<th></th>
<th>$|e|_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>$3.72 \times 10^{-6}$</td>
</tr>
<tr>
<td>Case 5</td>
<td>$5.46 \times 10^{-3}$</td>
</tr>
<tr>
<td>Case 12</td>
<td>$1.2 \times 10^{-7}$</td>
</tr>
<tr>
<td>Case 17</td>
<td>$3.89 \times 10^{-5}$</td>
</tr>
</tbody>
</table>

In conclusion, from the above results (table II), we were able to show that the proposed on-line algorithm is capable of accurately reconstructing the nonlinear orbit and give a very close parameters estimate. The next step is to prove it in hardware.
CHAPTER VI

EXPERIMENTAL RESULTS

6.1 Introduction

To test the feasibility of the proposed algorithm, hardware simulation experiments have been conducted using a Spectra Quest simulator, which consist of two aluminum rotors, steel shaft, sensors, and electric motor as shown in Fig. 42. The goal is to prove the validity of the algorithm and to classify real-time vibration data into one of the fault categories and dynamic modes. To demonstrate the performance of the diagnosis algorithm, the rotor-stator process was simulated under different fault conditions, i.e., imbalance, rubbing with different material (plastic and rubber), rubbing with imbalance, and at different frequencies. By using the Simulator, it offered us a wide range of benefits in gaining the understanding of predictive maintenance and learning to recognize the signatures of various machine faults. Different types of studies can be carried out with application specific kits. Each kit is designed to fit on the basic simulator.

Machinery fault simulator from Spectra Quest Co. has been used to study the signatures of common machinery faults. The bench-top system has a spacious modular design featuring versatility, operational simplicity, and robustness. Each component is machined to high tolerances, so it can be operated without conflicting
vibration. Then, depending on the situation, we can introduce various faults either individually or jointly in a totally controlled environment.

To gain an in-depth understanding of different vibration signatures, we needed to conduct controlled experiments on a device that emulates real world machinery. While we may need to analyze single faults one at a time, there are many occasions when we need to study dynamic stiffness, resonance, and speed when these components interact. A thorough analysis of vibration usually requires studies of several defects simultaneously to gain an understanding of real world vibration spectra, and that's what we applied in our simulation. With the Simulator, you can develop the expertise required to diagnose industrial machinery problems in well-controlled experiments.

When a plant is running at a high level of productivity, it is virtually impractical to gain an understanding of the kinematics and dynamics of machinery without adversely affecting production and profits. Therefore, the simulator is essential in environments where it is unacceptable to shut down production machinery for training and experimentation.
6.2 *Hardware Setup*

Different experiments were conducted using Spectra Quest Machinery Fault Simulator (Appendix A) as shown in Fig. 42.

![Fig. 42 Spectra Quest Machinery Simulator](image)

The belts on the right, and the green/gold gearbox were removed.
The simulator consists of two aluminum rotors mounted on steel shaft, with rolling element bearing, as shown in Fig. 43.

![Two Aluminum Rotors](image)

**Fig. 43 Two Aluminum Rotors**

- Two proximity probes were used to measure the displacement of the shaft in x and y direction. The use of this type of probes avoided the magnetic field interference commonly encountered with small diameter shafts. AIP standard proximity probes, eddy current sensors, LVDT's, etc. may be mounted both vertically and horizontally, and adjustable in the axial direction for measurements at various locations along the rotor shaft.

![Proximity Probes](image)

**Fig. 44 Proximity Probes**
To show the effects of coupling stiffness on rotor dynamics and vibration signature of various defects, a shaft coupling kit was used. Fig. 45 shows different types of coupling kits. In our experiment the first one from the left was used.

Fig. 45 Shaft Coupling Kit

Each kit consists of one gear, one helical beam, one rubber, and one steel coupling. The use of such a coupling was to show and clarify the complexities of machinery shaft misalignment problems.

Fig. 46 shows the setup of the simulator with all the dimensions. A weighted mass was added to the edge of the rotor to induce an imbalance. Notice that the weighted mass was added only to the edge of the right rotor. The left rotor was used only to balance the shaft and to prevent it from breaking. Finally, a mechanical rubbing kit was used to simulate a mechanical rub with different materials under a variety of conditions. It consists of an arm pin where it can be fitted with different material, such as, rubber, plastic, and copper. It is used to induce the rubbing phenomena for different fault cases.

The following are some of the measurements:

- Rotor Mass = 1 lb.
- Weight Mass = 1.5 Ounce
- Rotor has 18 holes, 6" diameter with two rows of tapped holes at every 20° (with lip for introducing unbalance force). For more information about the simulator see appendix A.

A Stanford spectral analyzer (Appendix B) was used to collect the data. The data were stored in different files to be used in the signal processing analysis.

6.3 Test Conducted

Different sets of experiments were conducted to generate different fault conditions under different frequencies. Appendix "C" shows all the cases, which were performed. The cases can be categorized into four different sets of experiments:
a) Normal Case (no faults)

b) Imbalance (where a weight was applied to the edge of the rotor)

c) Rubbing (where different material being applied to the shaft of the simulator, using a rubbing kit)

d) Rubbing with Imbalance

All the cases were performed at different speeds (or at different frequencies). A proximity probe was used to measure the displacement. Since this sensor uses eddy current, the material of the object is limited to metal. The eddy current depends on the magnetic force at the metal surface, or the distance between the coil and the object, the sensor coil inductance \( L \) will change with the eddy current, which varies the terminal voltage of the resonant circuit as a function of the distance. Detecting this signal will give the gap from the sensor to the object.

Data was collected by measuring the displacement in both \( x \) and \( y \) direction. These data sets can be used in our mathematical model to regenerate different parameters, which are associated with different fault conditions, as shown in Fig. 47. These parameters can be categorized into different sets of fault conditions for further use.

The following is a simplified version of the block diagram in which it shows how the real data has been used to estimate the parameters, see Appendix D for more details.
The mathematical models derived in Chapter II are used here to build the experimental simulation for different fault scenarios. Hence, the framework proposed (on-line algorithm) in Chapter IV is implemented as described in section 4.3.2. This implementation includes parameter estimation, filtering, and the classification of the faults in different categories (cases). Although the simulations were a little different, the approach was very similar, see Fig. 47.

Fig. 48 is a Matlab simulink block diagram, where it shows the flow of the proposed algorithm. It has been used to carry out the simulation and the parameter estimation of the real data. Data, for both x and y, was collected using the prescribed sensors, and was saved in both u(1) and u(2), respectively, as vectors. In addition, approximations of the derivative were used to obtain the first and second derivative, see Fig. 47. The inputs were multiplied by a certain gain to keep it within a specified
range. The data have been passed to the S-functions to evaluate the parameters. The parameters then can be classified and stored into different categories for future machine fault detection (see Appendix D for more details).
Fig. 48  Block Diagram which has been Extracted from the Matlab to Show the Simulation Process
6.4 Experimental Results

To prove the feasibility of the algorithm, real data have been generated and used. Fig. 49, 49, and 50 show the orbit diagrams, and the effect of changes in system parameters on the dynamics of the system. To distinguish between the normal case (no fault) and a faulty case, three different cases are shown:

- First case was the Normal case at 84 Hz

![Diagram](image-url)

Fig. 49 Normal Case at 84 Hz, a) Original Real Data, and b) Simulated Data
• The second case was Rubbing with a Rubber Pin (at 32 HZ):

Fig. 50 Rub with a Rubber Pin (32 HZ). a) Original Real Data, and b) Simulated Data.
- The third case was Rubbing with a Rubber Pin and Imbalance (32 HZ):

Fig. 51 Rub with a Rubber Pin with an Imbalance (32 HZ). a) Original Real Data and b) Simulated Data
The complicated motion and the sensitivity of the dynamics to parameters in
the presence of rub impact are clearly indicated, which confirm the previous
simulation and experimental results.

All the above cases showed the robustness of the proposed approach. Each set
of figures showed how successful the algorithm was in reconstructing the parameter
set. Hence, the rotor estimator (bottom figure) was able to track the state trajectory of
the system (top figure) even though the parameters were nonlinear. Each case can be
categorized within a certain range. The stability and the convergence of the system
for the normal and fault conditions have been very successful as shown above.

More cases can be simulated and stored to present different fault cases. In real
world, the “Rotor_Estimator_c” can take real data as input and regenerate the
parameters to be classified into different fault cases. Then by using neural network,
Euclidean distance ...etc. fault can be identified, classified and corrected before any
catastrophic incident might happen.

From all the above results, we can conclude that our proposed parameter
estimation algorithm has demonstrated excellent performance in the detection,
diagnosis, and isolation of faults.
CHAPTER VII
CONCLUSION

7.1 Concluding Remarks

The main contribution of this work is the formulation of the problem of fault
detection and diagnosis for rotating machinery in a statistical model-based
framework. The framework provided the foundation for an effective on-line model-
based monitoring and diagnosis for rotating machinery. Nonlinear discontinuous
phenomenon such as rub impact has been considered. The proposed approach is
based on computationally simple and efficient algorithms for signal processing and
parameter identification. The advantages of the proposed approach over other
monitoring and diagnosis algorithms that are based on state estimation were outlined.
The main contributions of this work can be summarized as follows:

1) Setting up a framework for the development of fault detection and
diagnosis algorithms for rotating machinery.

2) Characterization of rotor fault classes in terms of variations in the
parameters of nonlinear discontinuous differential equations.

3) Effective vibration signal denoising algorithm using Wavelets
algorithms.

4) Simulations and experimental validation of the diagnosis algorithm.
5) Computationally efficient / practical paradigm.

This dissertation offered a practical fault detection methodology as a diagnostic tool in dynamic data of rotating machinery. A description of the forces produced by rubbing and how they can initiate rotor instability were studied. Among the factors leading to destructive instability of rotors in high speed, high performance turbo machinery is the excessive rubbing between rotating and stationary parts.

The contribution of this work was mainly to identify the fault and categorize it into different fault cases, using a few parameters. By doing so, industry will save the time and the money to detect faults "on-line" in their early stages. This eliminates the need of storing a huge bank of data, analyze it, run different algorithm on it to get the results.
7.2 Future Work

Extensions of this work can be done in the following areas:

1. Theoretical extension: enlarging the class of problems, which might include the derivation of different models by adopting multiple degrees of freedom. Hence, this could be developed to incorporate the dynamical models of other nonlinear phenomena that need to be detected within the fault detection framework. Also it allows for the application of this approach to large-scale rotating machinery.

2. Implementation extensions: implementation of the algorithm in real-time by testing the fault detection scheme on real power plant.

3. Integration extension: integration of this model-based scheme within an advanced fault detection and diagnosis system.
REFERENCES


APPENDICES
### A. Machine Specification

<table>
<thead>
<tr>
<th>Electrical:</th>
<th>DC Drive:</th>
<th>AC Drive:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive:</td>
<td>½ HP Variable Speed DC Motor</td>
<td>½ HP 3 Phase Variable Speed</td>
</tr>
<tr>
<td>AC</td>
<td>Motor (with local a/o Remote Ctrl.)</td>
<td></td>
</tr>
<tr>
<td>RPM:</td>
<td>Maximum 4000</td>
<td>to over 10,000</td>
</tr>
<tr>
<td>Range:</td>
<td>0 to 4000 user selection</td>
<td>0 to 10,000 user selection</td>
</tr>
<tr>
<td>Voltage:</td>
<td>115/230 VAC, Single phase. 60/50 Hz</td>
<td>115/230 VAC, Single phase. 60/50 Hz</td>
</tr>
<tr>
<td></td>
<td>60/50 Hz</td>
<td></td>
</tr>
</tbody>
</table>

Both AC and DC drives can also be obtained at 1 HP rating. A panel box provides an easy and safe access to all cables for motor current signature analysis.

### Mechanical:

<table>
<thead>
<tr>
<th>Shaft Diameter:</th>
<th>5/8 in., Steel; ¾ in. Steel optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bearings:</td>
<td>2. Sealed Ball bearings</td>
</tr>
<tr>
<td>Bearing Housings:</td>
<td>2, aluminum Horizontally split bracket for simple and easy changes, tapped to accept transducer mount</td>
</tr>
<tr>
<td>Bearing Housing Base:</td>
<td>Completely movable using Jack Bolts for easy misalignment in all three planes</td>
</tr>
<tr>
<td>Axial Loading:</td>
<td>Easy to introduce using Jack Bolts</td>
</tr>
<tr>
<td>Rotors:</td>
<td>2, Aluminum, 6&quot; diameter with two rows of tapped holes at every 20° (with lip for introducing unbalance force)</td>
</tr>
</tbody>
</table>
Sheaves: Double V-Belt (finished bore and split tapered bushing)
Tensioners: Positive adjusting lever and sliding base
Gearbox: Three way straight tooth profile bevel gearbox with 1.5:1 ratio, accessible internals
Magnetic Brake: Manually adjustable, .5 - 10 lb.-in; designed to introduce desired load on the gearbox

Reciprocating Mechanism:
  * Strokes: 1.0" (25.4mm), 1.5" (37.5mm), and 2.0" (50.8mm)
  * Resistance Force: Preload adjustable using 3 coil springs

Mounting base and general structure: Precision Machined Aluminum
Foundation: ½ in. die cast aluminum base with removable stiffener and 6 Rubber isolators
Safety Cover: Clear, impact resistance hinged cover with safety cut-off interlock

Physical:

Operating weight: Approximately 130 lb. Size: L =37". W =20", H = 18.5"
Shipping Container: Reusable plywood
B. Spectral Analyzer Specification

The analyzer that has been used to collect the data has the following specification:

- SRS Stanford Research System Model SR785

- 2 Channels Dynamic Signal Analyzer

- CH 1 / CH2 A & B are voltage Inputs with 1M ohms, 50 Pf input impedance

- Setup had the following parameters:
  - Span = 400 Hz
  - Line Width = 1 Hz
  - Acquisition Time = 1 Sec
  - FFT line = 400
  - Base Freq = 1024 KHz
  - Start Freq = 0, Middle Freq= 200, End Freq = 400 Hz
  - Ch1 = X and CH2 = Y
C. **Different Cases for the Experiment**

- **Normal Case**:
  
<table>
<thead>
<tr>
<th>Freq</th>
<th>File: SCRNxxxx (X, Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 Hz</td>
<td>2, 3 (Plot File = 36)</td>
</tr>
<tr>
<td>32 Hz</td>
<td>4.5 (Plot File = 37)</td>
</tr>
<tr>
<td>42 Hz</td>
<td>6.7 (Plot File = 38)</td>
</tr>
<tr>
<td>52 Hz</td>
<td>8.9 (Plot File = 39)</td>
</tr>
<tr>
<td>62 Hz</td>
<td>10.11 (Plot File = 40)</td>
</tr>
<tr>
<td>72 Hz</td>
<td>12.13 (Plot File = 41. 42 rescaled )</td>
</tr>
<tr>
<td>84 Hz</td>
<td>14.15 (Plot File = 43)</td>
</tr>
</tbody>
</table>

- **Imbalance**:
  
<table>
<thead>
<tr>
<th>Freq</th>
<th>File: SCRNxxxx (X, Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 Hz</td>
<td>17.18 (Plot File = 31)</td>
</tr>
<tr>
<td>32 Hz</td>
<td>19.20 (Plot File = 32)</td>
</tr>
<tr>
<td>42 Hz</td>
<td>21.22 (Plot File = 33)</td>
</tr>
<tr>
<td>52 Hz</td>
<td>23.24 (Plot File = 34)</td>
</tr>
<tr>
<td>62 Hz</td>
<td>25.26 (Plot File = 35 @ 59.4 Hz)</td>
</tr>
<tr>
<td>Repeat</td>
<td>42 Hz</td>
</tr>
<tr>
<td>52 Hz</td>
<td>29.30</td>
</tr>
</tbody>
</table>

- **Rubbing (Plastic Pin)**:
  
<table>
<thead>
<tr>
<th>Freq</th>
<th>File: SCRNxxxx (X, Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 Hz</td>
<td>44.45 (Plot File = 58)</td>
</tr>
<tr>
<td>32 Hz</td>
<td>46.47 (Plot File = 59)</td>
</tr>
<tr>
<td>42 Hz</td>
<td>48.49 (Plot File = 60)</td>
</tr>
<tr>
<td>52 Hz</td>
<td>50.51 (Plot File = 61)</td>
</tr>
<tr>
<td>62 Hz</td>
<td>52.53 (Plot File = 62)</td>
</tr>
<tr>
<td>72 Hz</td>
<td>54.55 (Plot File = 63)</td>
</tr>
<tr>
<td>84 Hz</td>
<td>56.57 (Plot File = 64)</td>
</tr>
</tbody>
</table>

- **Rubbing (Plastic Pin) With Imbalance**:
  
<table>
<thead>
<tr>
<th>Freq</th>
<th>File: SCRNxxxx (X, Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 Hz</td>
<td>65.66 (Plot File = 75)</td>
</tr>
<tr>
<td>32 Hz</td>
<td>67.68 (Plot File = 76)</td>
</tr>
<tr>
<td>42 Hz</td>
<td>69.70 (Plot File = 77)</td>
</tr>
<tr>
<td>52 Hz</td>
<td>71.72 (Plot File = 78)</td>
</tr>
<tr>
<td>58 Hz</td>
<td>73.74 (Plot File = 79)</td>
</tr>
</tbody>
</table>
- **Rubbing (Rubber Pin):**

<table>
<thead>
<tr>
<th>Freq</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 HZ</td>
<td>80.81 (Plot File = 93)</td>
</tr>
<tr>
<td>32 HZ</td>
<td>82.83 (Plot File = 94)</td>
</tr>
<tr>
<td>42 HZ</td>
<td>84.85 (Plot File = 95)</td>
</tr>
<tr>
<td>52 HZ</td>
<td>86.87 (Plot File = 96)</td>
</tr>
<tr>
<td>62 HZ</td>
<td>88.89 (Plot File = 97)</td>
</tr>
<tr>
<td>72 HZ</td>
<td>90.91 (Plot File = 98)</td>
</tr>
<tr>
<td>84 HZ</td>
<td>92, ** (Plot File = 99)</td>
</tr>
</tbody>
</table>

- **Rubbing (Rubber Pin) With Imbalance:**

<table>
<thead>
<tr>
<th>Freq</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.05 HZ</td>
<td>100,101</td>
</tr>
<tr>
<td>32 HZ</td>
<td>102,103</td>
</tr>
<tr>
<td>42 HZ</td>
<td>104,105</td>
</tr>
<tr>
<td>52 HZ</td>
<td>106,107</td>
</tr>
<tr>
<td>58 HZ</td>
<td>108,109</td>
</tr>
</tbody>
</table>
D. Simulation Programs

* Main Program
  * Driver for RotDyn Dynamics of a Rotating Machine

```matlab
scenario = input('Enter case number [1-18]:');
t0 = 0;
tfinal = 150*pi;
SensorNoiseLevel = 1;
SteadyStateRange = 0.7;

PoincareSamplingRate = 1000;
deltat = 2*pi/PoincareSamplingRate; % divided by 1000 to get details
tspan = t0:deltat:tfinal;

[y0 = 0.01* rand(4,1);
y0 = [0.0 0.0 0 0.0]'];

options = odeset('RelTol',1e-4,'AbsTol',1e-6);
[t,y] = ode45('RotorDynamic', tspan, y0, options); % Solve differential Eqn, medium order method
SSTime = round(SteadyStateRange*length(tspan)/PoincareSamplingRate);
SSTime = SSTime*PoincareSamplingRate;
% Add sensor noise
z(:,1) = z(:,1) + randn(1,length(z(:,1)))*SensorNoiseLevel; % Add noise to X
z(:,3) = z(:,3) + randn(1,length(z(:,3)))*SensorNoiseLevel; % Add noise to Y

subplot(2,2,1); plot(t(SSTime:length(tspan)), y(SSTime:length(tspan),1)); title('a) Time history of X');
subplot(2,2,2); plot(t(SSTime:length(tspan)), y(SSTime:length(tspan),3)); title('b) Time history of Y');
subplot(2,2,3);
plot(y(SSTime:length(y(:,1)),1), y(SSTime:length(y(:,3)),3),'.');
title('c) State Trajectory or "Orbit Plot"');
subplot(2,2,4); plot(y(SSTime:PoincareSamplingRate:length(y(:,1)),1), y(SSTime:PoincareSamplingRate:length(y(:,3)),3),'.'); title('d) phase portrait');

input('Press Enter key to continue ...to plot the noisy DATA');

subplot(2,2,1); plot(t(SSTime:length(tspan)), z(SSTime:length(tspan),1)); title('a) Time history of X with noise');
```

subplot(2,2,2);plot(t(SSTime:length(tspan)),z(SSTime:length(tspan),3)), title('b) Time history of Y with noise'); subplot(2,2,3); plot(z(SSTime:length(z(:,1)),1),z(SSTime:length(z(:,3)),3),'.'); title(' State Trajectory or "Orbit Plot"') subplot(2,2,4);plot(z(SSTime:PoincareSamplingRate:length(z(:,1)),1), z(SSTime:PoincareSamplingRate:length(z(:,3)),3),'.'); title(' phase portrait')

input('Press Enter key to continue ...to calculate FFT');

ftx=fft(z(SSTime:PoincareSamplingRate:length(z(:,1))).1.1024); magftx = abs(ftx);
plot(magftx), title('FFT Magnitude'); fs=PoincareSamplingRate

xndof=z(:,1)'
yn dof=z(:,3)'
save xndof xndof; % Saving noisy X
save yndof yndof; % Saving noisy Y

xdof=y(:,1)'
ydof=y(:,3)';
save x dof xdo f; % Saving X
save ydof ydof; % Saving Y

input('Press Enter key to continue ...Equation time history of X & Y');

ls1= length(xndof);
ls2= length(yndof);

[C1,L1]=wavedec(xndof,6,'db3'); % Perform a level 3 decomposition
[C3,L3]=wavedec(yndof,6,'db3'); % Perform a level 3 decomposition

C1_A3=appcoef(C1,L1,'db3',6); % Extract level 3 approximation coefficients
C3_A3=appcoef(C3,L3,'db3',6); % Extract level 3 approximation coefficients

z1_A3=wrcoef('a',C1,L1,'db3',6); % reconstruct level 3 approximation
z3_A3=wrcoef('a',C3,L3,'db3',6); % reconstruct level 3 approximation

% Comparing 'x' with clean 'x'
subplot(2,2,1);plot(t(SSTime:length(tspan)),y(SSTime:length(xdof),1)); title('a)Time history of ' "X""); subplot(2,2,2);
plot(t(SSTime:length(tspan)),z1_A3(SSTime:ls1)),title('b) Approximation of ' "X"");

% Comparing 'y' with cleaned 'y'
subplot(2,2,3);plot(t(SSTime:length(tspan)),y(SSTime:length(ydof),3)); title('c) Time history For ' "Y"");
```matlab
subplot(2,2,4); plot(t(SSTime:length(tspan)),z3_A3(SSTime:ls2)), title('d) Approximation of "Y"')

% Comparing 'x vs y' with cleaned 'x vs y'
input('Press Enter key to continue ...for Orbit Diagram');
subplot(2,2,3);
plot(y(SSTime:length(xdof),1),y(SSTime:length(ydof),3),'o'), title('e) State Trajectory or "Orbit Plot"')
subplot(2,2,4); plot(z1_A3(SSTime:ls1),z3_A3(SSTime:ls2),'.'), title('f) Approximation of State Trajectory')

% Another method: Removing noise by thresholding
input('Press Enter key to continue ...for another method');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [thr,sohr,keepapp]=ddencmp('den', 'wv', xdof);
% clean_x=wdencmp('gbl',C1,L1,'db3',3,thr,sohr,keepapp);
% [thr,sohr,keepapp]=ddencmp('den', 'wv', ydof);
% clean_y=wdencmp('gbl',C3,L3,'db3',3,thr,sohr,keepapp);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if many trials are necessary, it is better to perform
decomposition one time and threshold it many times :
% decomposition.
[C1,L1] = wavedec(xdof,6,'sym8');
% threshold the decomposition structure [C1,L1].
clean_x = wden(C1,L1,'minimaxi','s','sln',6,'sym8');

[C2,L2] = wavedec(ydof,6,'sym8');
% threshold the decomposition structure [C2,L2].
clean_y = wden(C2,L2,'minimaxi','s','sln',6,'sym8');

% Comparing 'x' with De_noised 'x'
subplot(2,2,1);plot(t(SSTime:length(tspan)),y(SSTime:length(xdof),1)), title('a) Time history of "X"');
subplot(2,2,2);
plot(t(SSTime:length(tspan)),clean_x(SSTime:length(clean_x))),title('b) Approximation of "X"')

% Comparing 'y' with De_noised 'y'
subplot(2,2,3);plot(t(SSTime:length(tspan)),y(SSTime:length(ydof),3)), title('c) Time history For "Y"');
subplot(2,2,4);
plot(t(SSTime:length(tspan)),clean_y(SSTime:length(clean_y))), title('d) Approximation of "Y"')
```
Comparing 'x vs y' with De_noised 'x vs y'
input('Press Enter key to continue for Orbit plot...');
subplot(2,2,3);
plot(y(SSTime:length(xdof),1),y(SSTime:length(ydof),3),'.'), title('State Trajectory or "Orbit Plot"
subplot(2,2,4);
plot(clean_x(SSTime:length(clean_x)),clean_y(SSTime:length(clean_y))
',.'), title('Approximation of State Trajectory')
plot(clean_x(SSTime:length(clean_x)),clean_y(SSTime:length(clean_y)
',.'), title('Approximation of State Trajectory'),axis([-1.5 1.5 -1.5 1.5])
error_x = sqrtm((clean_x-xdof)'*(clean_x-xdof).')/ length(clean_x);
error_y = sqrtm((clean_y-ydof)'*(clean_y-ydof).')/ length(clean_y);
save clean_x clean_x;
save clean_y clean_y;
% Rotor Dynamics to represent the 18 different cases

function Xdot = RotorDynamic(t,X)
% RotDyn Dynamics of a Rotating Machine

global scenario
M=1;
Ks = 100000.0;
omega_s=sqrt(Ks/M);

diam=.03; % Disk Diameter
clrnc =0.00072; % Clearance
rho = diam/(2*clrnc); % Normalized Disc Radius
NoiseLevel=0.0;

%case 1:
if (scenario == 1)
    omega=1.0;
    E=0.4;
    mu=0.13;
    zeta=0.2;
    Kb=1.61e4;
    G=1.58;
end

%case 2
if (scenario == 2)
    omega=1.0;
    E=0.31;
    mu=0.68;
    zeta=0.35;
    Kb=1.22e4;
    G=1.2;
end

%case 3
if (scenario == 3)
    omega=1.0;
    E=0.31;
    mu=0.13;
    zeta=0.2;
    Kb=1.25e4;
    G=1.23;
end

%case 4
if (scenario == 4)
    omega=1.0;
    E=0.25;
    mu=0.13;
    zeta=0.2;
    Kb=1.0e4;
    G=0.98;
end

%case 5

if (scenario == 5)
    omega=1.0;
    E=0.45;
    mu=0.3;
    zeta=0.14;
    Kbeta=1.33e4;
    G=0.131;
end

%case 6
if (scenario == 6)
    omega=1.0;
    E=0.8;
    mu=0.1;
    zeta=0.032;
    Kbeta=1.e5;
    G=0.981;
end

%case 7
if (scenario == 7)
    omega=1.0;
    E=0.347;
    mu=0.13;
    zeta=0.024;
    Kbeta=1.389e4;
    G=0.136;
end

%case 8
if (scenario == 8)
    omega=1.0;
    E=0.133;
    mu=0.1;
    zeta=0.027;
    Kbeta=1.67e4;
    G=1.164;
end

%case 9
if (scenario == 9)
    omega=3.0;
    E=0.32;
    mu=0.13;
    zeta=0.135;
    Kbeta=1.124e5;
    G=1.1;
end

%case 10
if (scenario == 10)
    omega=3.0;
    E=0.74;
mu=0.15;
zeta=0.23;
Kbeta=2.68e4; \* 2.36 created similar but more contacts/rub points
G=.028;
end

%case 11
if (scenario == 11)
omega=3.0;
E=0.74;
mu=0.15;
zeta=0.27;
Kbeta=4.08e4;
G=.04;
end

%case 12
if (scenario == 12)
omega=1.0;
E=0.49;
mu=0.5;
zeta=0.14;
Kbeta=1.33e3;
G=.13;
end

%case 13
if (scenario == 13)
omega=1.0;
E=0.75;
mu=0.3;
zeta=0.16;
Kbeta=1.25e4;
G=.123;
end

%case 14
if (scenario == 14)
omega=3.0;
E=0.55;
mu=0.7;
zeta=0.79;
Kbeta=1.0e5;
G=.981;
end

%case 15
if (scenario == 15)
omega=3.0;
E=0.61;
mu=0.4;
zeta=0.79;
Kbeta=1.110e5;
G=1.09;
end
%case 16
if (scenario == 16)
    omega=1.0;
    E=0.7;
    mu=0.1;
    zeta=0.16;
    Kb=1.250e5;
    G=1.226;
end

%case 17
if (scenario == 17)
    omega=1.0;
    E=0.32;
    mu=0.1;
    zeta=0.027;
    Kb=4.0e4;
    G=.392;
end

%case 18
if (scenario == 18)
    omega=1.0;
    E=0.4;
    mu=0.1;
    zeta=0.027;
    Kb=5.0e4;
    G=.491;
end

theta=atan(X(3)/X(1));
R=sqrt(X(1)^2+X(3)^2);
delta= R-1;

phi = X(1)*X(4)-X(3)*X(2); % in the code (results match those in Figure 2.5)

phi = phi/(R*R*omega_s);
psi=phi*R+omega*rho;
SignPsi=sign(psi);

Xd(1)=X(2);
Xd(2)=2*zeta*X(2) -X(1) + E*omega*omega*cos(omega*t);
if (delta>0)
    Xdot(2) = Xdot(2) - Kb*(delta^1.5)*(X(1)/R-SignPsi*mu*X(3)/R) + NoiseLevel*randn;
end

Xd(3)=X(4);
Xd(4)=2*zeta*X(4) -X(3) + E*omega*omega*sin(omega*t)-G + NoiseLevel*randn;
if (delta>0)
Xdot(4) = Xdot(4) - Kbeta*(delta^1.5)*(X(3)/R+SignPsi*mu*X(1)/R);
end
Xdot = Xdot';

end
\% IDDriver.m

global Param;
global FilteredSignal;

\% Param0 is random values
Param0(1) = randperm(1); \% Omega
Param0(1) = .1+.7*rand; \% E
Param0(2) = .1+.8*rand; \% \mu
Param0(3) = .032 + .8*rand; \% \beta
Param0(4) = 5-4*rand; \% kbeta/\epsilon
Param0(5) = .1+1.9*rand; \% G

load clean_x;
load clean_y;

FilteredSignal = clean_x'; \% (clean'_x' clean'_y')

\% options(2) = 1e-4;
\% options(3) = 1e-4;
\% Param = LEASTSQ('IdModelError',Param0, options, 'IdModelErrorGrad')
\% [Param,options,F,J] = LEASTSQ('IdModelError',Param0)

\% IdModelErrorGrad.m

function MSEErrorGrad = IdModelErrorGrad(Param)
    global y;
    MSEErrorGrad = [-y(:,2) -y(:,4)];
end

\% Xdot = IdRotorDynamic(t,X)

\% Rotor Dynamic of a Rotating Machine

M=1;
Ks = 100000.0;
omega_s = sqrt(Ks/M);

\% diam = 0.03; \% Disk Diameter
clrlnc = 0.00072; \% Clearance
rho = diam/(2*clrlnc); \% Normalized Disc Radius
NoiseLevel = 0.0;

theta = atan(X(3)/X(1));
R = sqrt(X(1)^2*X(1)+X(3)^2*X(3));
delta = R-1;

phi = X(1)*X(2)-X(3)*X(2); \% in the code (results match those in Figure 2.5)
phi = phi/(R*R*omega_s);
psi=phi*R+omega*rho;
SignPsi=sign(psi);

Xdot(1)=X(2);
Xdot(2)=-2*zeta*X(2) -X(1) + E*omega*omega*cos(omega*t);
if (delta>=0)
  Xdot(2) = Xdot(2) - Kbeta*(delta^1.5)*(X(1)/R-SignPsi*mu*X(3)/R) + NoiseLevel*randn;
end

Xdot(3)=X(4);
Xdot(4)=-2*zeta*X(4) -X(3) + E*omega*omega*sin(omega*t)-G +
NoiseLevel*randn;
if (delta>=0)
  Xdot(4) = Xdot(4) - Kbeta*(delta^1.5)*(X(3)/R+SignPsi*mu*X(1)/R);
end
Xdot = Xdot';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'
IDDriver.m

global Param;
global FilteredSignal;

\ Param0(1)= randperm(1); \ Omega
Param0(1)= .1+.7*rand; \ E
Param0(2)= .1+.8*rand; \ mu
Param0(3)= .032 + .8*rand; \ zeta
Param0(4)= 5-4*rand; \ %zeta/le4
Param0(5)= .1+1.9*rand; \ G

load clean_x;
load clean_y;

FilteredSignal= clean_x'; \ clean_x' clean_y';

\ options(2) = 1e-4;
\ options(3) = 1e-4;
\ Param=LEASTSQ('IdModelError',Param0, options, 'IdModelErrorGrad')
[Param,options,F,J]=LEASTSQ('IdModelError',Param)
*************** RotorParam.m, Recursive method***************

function Sdot = IdRotorGradient(t, S)

% RotDyn Dynamics of a Rotating Machine

global CSU_omega;
global CSU_E;
global CSU_Mu;
global CSU_zeta;
global CSU_Kbeta;
global CSU_G;
global y;
global k;

% [CSU_zeta, CSU_Kbeta/10e4, CSU_Mu, CSU_E, CSU_omega, CSU_G]

M=1;
Ks = 100000.0;
CSU_omega_s=sqrt(Ks/M);
alpha=1.5;

diam=.03; % Disk Diameter
clrnc =0.00072; % Clearance
rho = diam/(2*clrnc); % Normalized Disc Radius
NoiseLevel=0.0;

if abs(S(25))<1e-12
   S(25)=1e-12
end

if abs(S(25))<1e-12 & abs(S(27))<1e-12
   theta = 0;
else
   if abs(S(25))<1e-12 & abs(S(27))>1e-12
      theta = sign(S(27))*pi/2;
   else
      theta=atan(S(27)/S(25));
   end
end

R=sqrt(S(25)*S(25)+S(27)*S(27));
delta= R-1;

if(R>0)
   phi = S(25)*S(28)-S(27)*S(26); % in the code (results match those in Figure 2.5)
   phi = phi/(R*R*CSU_omega_s);
   psi=phi*R*CSU_omega*rho;
   SignPsi=sign(psi);
else
   SignPsi=0
end

if(delta>=0)
   if abs(S(25))>1e-12
      ThetaXX=-(S(27)*cos(theta)^2)/S(25)^2;
      ThetaXX=ThetaXX*(-sin(theta)-SignPsi*CSU_Mu*cos(theta));
      ThetaXY=(cos(theta)^2)/S(25);
      ThetaXY=ThetaXY*(-sin(theta)-SignPsi*CSU_Mu*cos(theta));
      ThetaYY=(cos(theta)^2)/S(25);
      ThetaYY=ThetaYY*(cos(theta)-SignPsi*CSU_Mu*sin(theta));
   end
end
ThetaXY = -(S(27) * cos(theta)^2) / S(25)^2;
ThetaXY = ThetaXY * (cos(theta) - SignPsi * CSU_Mu * sin(theta));
else
    ThetaXX = 0
    ThetaYY = 0
end

DeltaX = alpha * cos(theta) * (delta^2 * (alpha - 1));
DeltaY = alpha * sin(theta) * (delta^2 * (alpha - 1));
GammaXX = CSU_Kbeta * (DeltaX * cos(theta) -
SignPsi * CSU_Mu * sin(theta)) + (delta * alpha) * ThetaXX;
GammaXY = CSU_Kbeta * (DeltaX * cos(theta) -
SignPsi * CSU_Mu * sin(theta)) + (delta * alpha) * ThetaXY;

GammaYY = CSU_Kbeta * (DeltaX * sin(theta) + SignPsi * CSU_Mu * cos(theta)) +
(delta * alpha) * ThetaYY;

GammaXY = CSU_Kbeta * (DeltaX * sin(theta) + SignPsi * CSU_Mu * cos(theta)) +
(delta * alpha) * ThetaXY;
end

% ------------------------ CSU_zeta ------------------------
Sdot(1) = S(2);
Sdot(2) = -2 * CSU_zeta * S(2) - S(1) - 2 * S(26);
if (delta >= 0)
    Sdot(2) = Sdot(2) - GammaXX * S(1) - GammaXY * S(3);
end

Sdot(3) = S(4);
Sdot(4) = -2 * CSU_zeta * S(4) - S(3) - 2 * S(28);
if (delta >= 0)
    Sdot(4) = Sdot(4) - GammaYY * S(3) - GammaXY * S(1);
end

% ------------------------ CSU_Kbeta ------------------------
Sdot(5) = S(6);
Sdot(6) = -2 * CSU_zeta * S(6) - S(5);
if (delta >= 0)
    Sdot(6) = Sdot(6) - ((delta * alpha) * (cos(theta) -
SignPsi * CSU_Mu * sin(theta)) + GammaXX * S(5));
end

Sdot(7) = S(8);
Sdot(8) = -2 * CSU_zeta * S(8) - S(7);
if (delta >= 0)
    Sdot(8) = Sdot(8) -
((delta * alpha) * (sin(theta) + SignPsi * CSU_Mu * cos(theta)) + GammaYY * S(7));
end

% ------------------------ CSU_Mu ------------------------
Sdot(9) = S(10);
Sdot(10) = -2 * CSU_zeta * S(10) - S(9);
if (delta >= 0)
    Sdot(10) = Sdot(10) -
CSU_Kbeta * (delta * alpha) * SignPsi * sin(theta) + GammaXX * S(9));
end

Sdot(11) = S(12);
Sdot(12) = -2 * CSU_zeta * S(12) - S(11);
if (delta >= 0)
Sdot(12) = Sdot(12) - (CSU_kbeta*(delta^alpha)*SignPsi*cos(theta)+GammaYY*S(11));
end

% ------------------------ CSU_E ------------------------
Sdot(13)=S(14);
Sdot(14)=2*CSU_zeta*S(14) - S(13)+CSU_omega*CSU_omega*cos(CSU_omega*t);
if (delta>=0)
  Sdot(14) = Sdot(14)-GammaXX*S(13);
end

Sdot(15)=S(16);
Sdot(16)=2*CSU_zeta*S(16) - S(15)+CSU_omega*CSU_omega*sin(CSU_omega*t);
if (delta>=0)
  Sdot(16) = Sdot(16)-GammaYY*S(15);
end

% ------------------------ CSU_omega ------------------------
Sdot(17)=S(18);
Sdot(18)=2*CSU_zeta*S(18) - S(17)+CSU_E*(2*CSU_omega*cos(CSU_omega*t)-
        CSU_omega*CSU_omega*t*sin(CSU_omega*t));
if (delta>=0)
  Sdot(18) = Sdot(18)-GammaXX*S(17);
end

Sdot(19)=S(20);
Sdot(20)=2*CSU_zeta*S(20) - S(19)+CSU_E*(2*CSU_omega*sin(CSU_omega*t)+CSU_omega*CSU_omega*t*cos(
        CSU_omega*t));
if (delta>=0)
  Sdot(20) = Sdot(20)-GammaYY*S(19);
end

% ------------------------ CSU_G ------------------------
Sdot(21)=S(22);
Sdot(22)=0;
if (delta>=0)
  Sdot(22) = Sdot(22);
end

Sdot(23)=S(24);
Sdot(24)=2*CSU_zeta*S(24) -S(23)-1;
if (delta>=0)
  Sdot(24) = Sdot(24)-GammaYY*S(23);
end

% ------------------------ model dynamics (x,xdot,y,ydot) ------------------------
Sdot(25)=S(26);
Sdot(26)=2*CSU_zeta*S(26) -S(25) +
        CSU_E*CSU_omega*CSU_omega*cos(CSU_omega*t);
if (delta>=0 & R>0)
  Sdot(26) = Sdot(26) - CSU_Kbeta*(delta^1.5)*S(25)/R-
        SignPsi*CSU_Mu*S(27)/R;
end
Sdot(27) = S(28);
Sdot(28) = -2*CSU_zeta*S(28) - S(27) + 
CSU_E*CSU_omega*CSU_omega*sin(CSU_omega*t) - CSU_G;
if (delta>0 & R>0)
    Sdot(28) = Sdot(28) - 
CSU_Kbeta*(delta^1.5)*(S(27)/R+SignPsi*CSU_Mu*S(25)/R);
end
Sdot = Sdot';

% -----------------------------------

%Sdot = Sdot';

global Par_der_x;      % partial Derivative with respect to x
global par_der_y;      % partial Derivative with respect to y
global delta_alpha_x;  % partial Derivative of delta with respect to x
global delta_alpha_y;  % partial Derivative of delta with respect to y
global alpha;
global delta;
global sin_theta;
global cos_theta;
global mu;
global omega;

global Par_x_zeta;
global Par_y_zeta;
global Par_x_Kbeta;
global Par_y_Kbeta;
global Par_x_Mu;
global Par_y_Mu;
global Par_x_E;
global Par_y_E;
global Par_x_omega;
global Par_y_omega;
global Par_x_G;
global Par_y_G;

Bxk=100;
Byk=100;
omega=1;
% Param0(1) = randperm(1);  % Omega
Param(1) = .1+.7*rand;       % E
Param(2) = .1+.8*rand;       % mu
Param(3) = .032 + .8*rand;   % zeta
Param(4) = 5-4*rand;         % kbeta/le4
Param(5) = .1+1.9*rand;      % G

load clean_x;
load clean_y;

% define
Cos_sin= cos_theta-mu*sin_theta;
sin_cos= sin_theta + mu*cos_theta;
denom_x=K_beta*{{(RotorDeriv(:,3))*(Cos_sin) + ((delta^alpha)*
*RotorDeriv(:,1))}};
denom_y=K_beta*{{(RotorDeriv(:,4))*(sin_cos) + ((delta^alpha)*
*RotorDeriv(:,2))}};

% partial Derivative with respect to ZETA
Par_x_zeta=-(2*x(2)) / denom_x;
Par_y_zeta=-(2*x(4)) / denom_y;

% gradient descend approach
param(3)=param(3)-Bxk*Par_x_zeta*(xdof-clean_x)-
Bxy*Par_y_zeta*(ydof-clean_y);

% partial Derivative with respect to K_BETA
Par_x_Kbeta=(((delta^alpha)*(Cos_sin)) / denom_x;
Par_y_Kbeta=(((delta^alpha)*(sin_cos)) / denom_y;

% gradient descend approach to k_beta
param(4)=param(4)-Bxk*Par_x_Kbeta*(xdof-clean_x)-
Bxy*Par_y_Kbeta*(ydof-clean_y);

% partial Derivative with respect to Mu
Par_x_Mu=(((delta^alpha)*(sin_theta)) / denom_x;
Par_y_Mu=(((delta^alpha)*(cos_theta)) / denom_y;

% gradient descend approach to mu
param(2)=param(2)-Bxk*Par_x_Mu*(xdof-clean_x)-Bxy*Par_y_Mu*(ydof-
clean_y);

% partial Derivative with respect to E
Par_x_E=-(omega^2*cos(omega*t)) / denom_x;
Par_y_E=-(omega^2*sin(omega*t))/ denom_y;

% gradient descend approach to E
param(1)=param(1)-Bxk*Par_x_E*(xdof-clean_x)-Bxy*Par_y_E*(ydof-
clean_y);

omega=1;
% partial Derivative with respect to omega
%Par_x_omega=(E*(2*omega*cos(omega*t)+ (omega^2 *sin(omega*t)))/
denom_x;
%Par_y_omega=(E*(2*omega*sin(omega*t) - (omega^2 *cos(omega*t)))/
denom_y;
%gradient descend approach to omega
%param(0)=param(0)-B*xk *Par_x_omega*(xdof-clean_x)-
B*y*Par_y_omega*(ydof-clean_y);

% partial Derivative with respect to G
Par_x_G=0;
Par_y_G=-1 / denom_y;
%gradient descend approach to omega
param(5)=param(5)-B*xk *Par_x_G*(xdof-clean_x)-B*y*Par_y_G*(ydof-
clean_y);
end

----------------------------------------------- MS Error-----------------------------------------------

function MSEerror = IdModelError(Param)

global FilteredSignal;
global omega;
global E;
global mu;
global zeta;
global Kbeta;
global G;
global y;

omega = 1;
E=Param(1);
mu=Param(2);
zeta=Param(3);
Kbeta=1.e4*Param(4);
G=Param(5);

t0 = 0;
tfinal = 5*pi;

% Added for Matlab 5.0
PoincareSamplingRate=1000;
deltat = 2*pi/PoincareSamplingRate; % divided by 1000 to get details
tspan=t0:deltat:tfinal;
y0 = 0.01* rand(4,1); %[0.0 0.0 0.0 0.0]'
[E,mu,zeta,Kbeta/1.e4,G]
options = odeset('RelTol',1e-4,'AbsTol',1e-6);
[t, y] = ode45('IdRotorDynamic',tspan,y0,options); %Solve differential
Eqn, medium order method
x dof = y(:,1);
y dof = y(:,3);
z = [x dof y dof];

% Loading the filtered signal

MSE error = (Filtered Signal - x dof);
% diag(sqrt(MSE Error * MSE Error))'

end

*****************************************************************************
*****RotorDeriv.m*****************************************************************************
function Derivative = RotorDeriv(t,X)

global Par der x; % partial Derivative with respect to x
global par der y; % partial Derivative with respect to y
global delta alpha x; % partial Derivative of delta with respect to x
global delta alpha y; % partial Derivative of delta with respect to y
global alpha;
global delta;
global clrnc;
global sin theta;
global cos theta;
global mu;
global omega;

alpha = 1.5;
clrnc = 0.00072;
mu = 1;
omega = 1;
theta = ATAN(y/x);
delta = ((sqrt(x^2+y^2)/c)-1);

sin theta = y/sqrt(x^2+y^2);
cos theta = x/sqrt(x^2+y^2);

par der x = (sin theta + mu * cos theta) * (y/x^2) * ((cos theta)^2); % pder (cos-mu*sin)
par der y = (cos theta - mu * sin theta) * ((cos theta)^2)/x; % pder (sin+mu*cos)

delta alpha x = (alpha/c) * cos theta * (delta^((alpha-1))); % pder (delta alpha of x)
delta alpha y = (alpha/c) * sin theta * (delta^((alpha-1))); % pder (delta alpha of y)

Derivative = [par der x par der y delta alpha x delta alpha y]';

end

*****************************************************************************
*****IDDRIVE.m*****************************************************************************

global Param;
global omega;
global E;
global Mu;
global zeta;
global KBeta;
global G;
global y;

alpha=1.5;
clrnc=0.00072;
t0 = 0;
tfinal = 5*pi;
Bxk=100;
Byk=100;
epsilon=.01;

PoincareSamplingRate=1000;
deltat = 2*pi/PoincareSamplingRate; % divided by 1000 to get details
tspan=t0:deltat:tfinal;

%*****************************************************************************

% Param0(1) = randperm(1); % Omega
Param1(0) = .1+.7*rand; % E
Param2(0) = .1+.8*rand; % mu
Param3(0) = .032 + .8*rand; % zeta
Param4(0) = 5-4*rand; % kbeta/1e4
Param5(0) = .1+1.9*rand; % G
k=1;

while epsilon > 0.01
  k=k-1;
  omega=1;
  E=Param1(k);
  mu=Param2(k);
  zeta=Param3(k);
  KBeta=1.4*Param4(k);
  G=Param5(k);

  %*****************************************************************************

  theta= ATAN(x(3)/x(1));
delta= ((sqrt(x(1)^2+x(3)^2)/clrnc)-1);

  sin_theta= x(3)/sqrt(x(1)^2+x(3)^2);
cos_theta= x(1)/sqrt(x(1)^2+x(3)^2);

  par_der_x=(sin_theta+ mu* cos_theta)*x(3)/(x(1)) + (((cos_theta)^2));% pder(cos+mu-sin)
  par_der_y=(cos_theta- mu* sin_theta)*(((cos_theta)^2))/x(1));% pder
  (sin+mu*cos)

  delta_alpha_x= (alpha/clrnc)*cos_theta* (delta^(alpha-1));% pder
  (delta_alpha of x)
delta_alpha_y = (alpha/clrnc)*sin_theta* (delta^(alpha-1));

% define
Cos_sin= cos_theta-mu*sin_theta;
sin_cos= sin_theta + mu*cos_theta;
denom_x=K_beta *((delta_alpha_x*(Cos_sin)) + ((delta^alpha)
*par_der_x));
denom_y=K_beta *((delta_alpha_y*(sin_cos)) + ((delta^alpha)
*par_der_y));

% load clean_x;
load clean_y;

% partial Derivative with respect to E
Par_x_E=-(omega^2 * cos(omega^t)) / denom_x;
Par_y_E=-(omega^2 * sin(omega^t))/ denom_y;
% gradient descend approach to E
param1(k+1)=param(k)-Bxk*Par_x_E*(xdof-clean_x)-By*Par_y_E*(ydof-
clean_y);
if abs(param1(k+1)-param1(k)) < epselon
end
E=param1(k+1)

% partial Derivative with respect to Mu
Par_x_Mu=((delta^alpha)(sin_theta)) / denom_x;
Par_y_Mu=((delta^alpha)(cos_theta)) / denom_y;
% gradient descend approach to mu
param2(k+1)=param2(k)-Bxk*Par_x_Mu*(xdof-clean_x)-
By*Par_y_Mu*(ydof-clean_y);
if abs(param2(k+1)-param2(k)) < epselon
end
Mu=param2(k+1);

% partial Derivative with respect to ZETA
Par_x_zeta=-(2*x(2)) / denom_x;
Par_y_zeta=-(2*x(4)) / denom_y;

% gradient descend approach
param3(k+1)=param3(k)-Bxk*Par_x_zeta*(xdof-clean_x)-
By*Par_y_zeta*(ydof-clean_y);
if abs(param3(k+1)-param3(k)) < epselon
end
zeta=param3(k+1);

% partial Derivative with respect to K_BETA
Par_x_Kbeta=((delta^alpha)(Cos_sin)) / denom_x;
Par_y_Kbeta=((delta^alpha)(sin_cos)) / denom_y;

%gradient descend approach to k_beta
param4(k+1)=param4(k)-Bxk*Par_x_Kbeta*(x dof-clean_x)-
Bxy*Par_y_Kbeta*(y dof-clean_y);

if abs(param4(k+1)-param4(k)) < epsilon
end
Kbeta=param4(k+1);

% partial Derivative with respect to omega=1
%Par_x_omega=(E*(2*omega*cos(omega*t)+ (omega^2 *sin(omega*t)))/
denom_x;
%Par_y_omega=(E*(2*omega*sin(omega*t)- (omega^2 *cos(omega*t)))/
denom_y;
%gradient descend approach to omega
%param(0)=param(0)-Bxk*Par_x_omega*(x dof-clean_x)-
Bxy*Par_y_omega*(y dof-clean_y);

% partial Derivative with respect to G
Par_x_G=0;
Par_y_G=-1 / denom_y;
%gradient descend approach to G
param5(k+1)=param(k)-Bxk*Par_x_G*(x dof-clean_x)-Bxy*Par_y_G*(y dof-
clean_y);

if abs(param5(k+1)-param5(k)) < epsilon
end
G=param5(k+1);

y0 = 0.01* rand(4,1); %[0.0 0.0 0 0.0]
[%[E,mu,zeta,Kbeta/1.e4,G]
options = odeset('RelTol',le-4,'AbsTol',le-6);
[t,y] = ode45('IdRotorDynamic',tspan,y0,options); %Solve differential
Equ, medium order method

x dof=y(:,1);
y dof=y(:,3);
z=[x dof y dof];

% Loading the filtered signal

%MSEerror= (FilteredSignal-x dof);
% diag(sqrt(MSEerror'*MSEerror))'
% ***************Recursive method, rotor_process_c***************

function [sys,x0,str,ts] = Rotor_Process_c(t,x,u,flag)

% The following outlines the general structure of an S-function.
% parameters=[a1 a2 a3 b1 b2];

switch flag,
    case 0,
        [sys,x0,str,ts]=mdlInitializeSizes;
    case 1,
        sys=mdlDerivatives(t,x,u);
    case 3,
        sys=mdlOutputs(t,x,u);
    case {2,4,9}
        sys = [];
    otherwise
        error(['Unhandled flag = ',num2str(flag)]);
end

end sfuntmpl

% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
function [sys,x0,str,ts]=mdlInitializeSizes
% call simsizes for a sizes structure, fill it in and convert it to a sizes array.
% Note that in this example, the values are hard coded. This is not a recommended practice as the characteristics of the block are typically defined by the S-function parameters.
sizes = Simsizes;

sizes.NumContStates = 4; % x, dx/dt, y, dy/dt
sizes.NumDiscStates = 0;
sizes.NumOutputs = 16; % x, dx/dt, y, dy/dt and Regressors (6-6)
sizes.NumInputs = 5; % a1, a2, a3, b1, b2
sizes.DirFeedthrough = 12; % Regressors for parameter estimator
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = Simsizes(sizes);

% initialize the initial conditions

x0 = 0.01 * rand(4, 1); % rand(4, 1);

% str is always an empty matrix
str = [];

% initialize the array of sample times
Ts = [0 0];

% endmdlInitializeSizes

% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step requirements.
function sys=mdlDerivatives(t,x,u)

a1 = u(1);
a2 = u(2);
a3 = u(3);
b1=u(4);
b2=u(5);

M=1;
Ks = 100000.0;
CSU_omega_s=sqrt(Ks/M);
CSU_omega=1;

diam=.03; // Disk Diameter
cldrnc =0.00072; // Clearance
rho = diam/(2*cldrnc); // Normalized Disc Radius
RubNoise=0.01;
LinearNoiseLevel=0.01;

if(x(1)<1e-12 & x(3)<1.e-12)
  theta = 0;
else
  if (x(1)<1e-12 & x(3)>1e-12)
    theta=pi/2;
  else
    theta=atan(x(3)/x(1));
  end
end

R=sqrt(x(1)*x(1)+x(3)*x(3));
delta= R-1;

if(R>0)
  phi = x(1)*x(4)-x(3)*x(2); // in the code results match those in Figure 2.5
  phi = phi/(R*R*CSU_omega_s);
  psi=phi*R+CSU_omega*rho;
  SignPsi=sign(psi);
else
  SignPsi=0;
end

phi_c=(delta^1.5)*x(1)/R;
phi_s=(delta^1.5)*x(3)/R;

xl=x(2);
x2=-x(1)-2*a1*x(2)+b1*cos(t)+ LinearNoiseLevel*randn;
  if (delta>0 & R>0)
    x2=x2-a2*phi_c + a3*SignPsi*phi_s+ LinearNoiseLevel*randn;
  end
x3=x(4);
x4=-x(3)-2*a1*x(4)+b1*sin(t)-b2+ RubNoise*randn;
  if (delta>0 & R>0)
    x4=x4-a2*phi_s - a3*SignPsi*phi_c+ RubNoise*randn;
  end

sys = [xl x2 x3 x4]';

end mdlUpdate
function sys=mdlOutputs(t,x,u)
%Calculate regressors as feed through output signals
global z;
a1=u(1);
a2=u(2);
a3=u(3);
b1=u(4);
b2=u(5);

M=1;
Ks = 100000.0;
CSU_omega_s=sqrt(Ks/M);
CSU_omega=1;

diam=.03;  % Disk Diameter
clrnc =0.00072;  % Clearance
rho = diam/(2*clrnc);  % Normalized Disc Radius
RubNoise=0.01;
LinearNoiseLevel=0.01;
Sens_Noise=0.1;

if(x(1)<e-12 & x(3)<1.e-12)
    theta = 0;
else
    if (x(1)<e-12 & x(3)>1e-12)
        theta=pi/2;
    else
        theta=atan(x(3)/x(1));
    end
end
R=sqrt(x(1)*x(1)+x(3)*x(3));
delta= R-1;

if(R>0)
    phi = x(1)*x(4)-x(3)*x(2);
    phi = phi/(R*R*CSU_omega_s);
    psi=phi*R+CSU_omega*rho;
    SignPsi=sign(psi);
else
    SignPsi=0;
end

phi_c=(delta^1.5)*x(1)/R;
phi_s=(delta^1.5)*x(3)/R;

v=zeros(6,1);
z=zeros(6,1);
\[ v(1) = -2 \times x(2); \]
\[ \text{if (delta>0 \& R>0)} \]
\[ v(2) = -\phi_c; \]
\[ v(3) = \text{SignPsi} \times \phi_s; \]
\[ \text{else} \]
\[ v(2) = 0.0; \]
\[ v(3) = 0.0; \]
\[ \text{end} \]
\[ v(4) = \cos(t); \]
\[ v(5) = 0; \]
\[ z(1) = -2 \times x(4); \]
\[ \text{if (delta>0 \& R>0)} \]
\[ z(2) = -\phi_s; \]
\[ z(3) = -\text{SignPsi} \times \phi_c; \]
\[ \text{else} \]
\[ z(2) = 0.0; \]
\[ z(3) = 0.0; \]
\[ \text{end} \]
\[ z(4) = \sin(t); \]
\[ z(5) = -1; \]

\[ v(6) = -x(1) - 2 \times a1 \times x(2) + b1 \times \cos(t) + \text{LinearNoiseLevel} \times \text{randn}; \]
\[ \text{if (delta>0 \& R>0)} \]
\[ \text{if (delta>0)} \]
\[ v(6) = v(6) - a2 \times \phi_c + a3 \times \text{SignPsi} \times \phi_s + \text{RubNoise} \times \text{randn}; \]
\[ \text{end} \]
\[ v(6) = -v(6); \]
\[ z(6) = -x(3) - 2 \times a1 \times x(4) + b1 \times \sin(t) - b2 \times \text{LinearNoiseLevel} \times \text{randn}; \]
\[ \text{if (delta>0 \& R>0)} \]
\[ \text{if (delta>0)} \]
\[ z(6) = z(6) - a2 \times \phi_s - a3 \times \text{SignPsi} \times \phi_c + \text{RubNoise} \times \text{randn}; \]
\[ \text{end} \]
\[ z(6) = -z(6); \]
\[ xn = x; \]
\[ xn(1) = xn(1) + \text{Sens Noise} \times \text{randn}; \]
\[ xn(3) = xn(3) + \text{Sens Noise} \times \text{randn}; \]

\[
[C1,L1]=\text{wavedec}(xn(1),3,'db3'); \quad \text{Perform a level 3 decomposition} \\
[C3,L3]=\text{wavedec}(xn(3),3,'db3'); \quad \text{Perform a level 3 decomposition} \\
\]
\[ \text{C1_A1} = \text{appcoef}(C1,L1,'db3',3); \quad \text{Extract level 3 approximation} \]
\[ \text{coefficients} \]
\[ \text{C3_A3} = \text{appcoef}(C3,L3,'db3',3); \quad \text{Extract level 3 approximation} \]
\[ \text{coefficients} \]
\[ xn(1) = \text{wrcoef}('a',C1,L1,'db3',3); \quad \text{reconstruct level 3} \]
\[ \text{approximation} \]
\[ xn(3) = \text{wrcoef}('a',C3,L3,'db3',3); \quad \text{reconstruct level 3 approximation} \]
\[ [C2,L2]=\text{wavedec}(v(6),3,'db3'); \quad \text{Perform a level 3 decomposition} \\
[C4,L4]=\text{wavedec}(z(6),3,'db3'); \quad \text{Perform a level 3 decomposition} \]
C2_A2 = appcoef(C2, L2, 'db3', 3); % Extract level 3 approximation coefficients
C4_A4 = appcoef(C4, L4, 'db3', 3); % Extract level 3 approximation coefficients
v6 = wrcoef('a', C2, L2, 'db3', 3); % Reconstruct level 3 approximation
z6 = wrcoef('a', C4, L4, 'db3', 3); % Reconstruct level 3 approximation

\sys = [x; Model_X; Model_Y];

\sys = [x; Model_X; Model_Y];
sys = [x; v; r; z];

% end mdlOutputs

%******************************************************************************
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time. Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%******************************************************************************
function sys = mdlGetTimeOfNextVarHit(t, x, u)

sampleTime = 1; % Example, set the next hit to be one second
% later.
sys = t - sampleTime;

% end mdlGetTimeOfNextVarHit

%******************************************************************************
% mdlTerminate
% Perform any end of simulation tasks.
%******************************************************************************
function sys = mdlTerminate(t, x, u)

sys = [];

% end mdlTerminate

******************************************************************************
function [sys, x0, str, ts] = Rotor_Estimator_C(t, x, u, flag)
% SFUNCTMP General M-file S-function template
% With M-file S-functions, you can define your own ordinary
differential equations (ODEs), discrete system equations, and/or
just about any type of algorithm to be used within a Simulink block

diagram.

The general form of an M-File S-function syntax is:

\[ \text{SYS, X0, STR, TS} = \text{SFUNC}(T, X, U, \text{FLAG}, P1, \ldots, Pn) \]

What is returned by SFUNC at a given point in time, \( T \), depends on the
value of the FLAG, the current state vector, \( X \), and the current
input vector, \( U \).

<table>
<thead>
<tr>
<th>FLAG RESULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>
| 0 \([\text{SIZEs}, \text{X0}, \text{STR}, \text{TS}]\) Initialization, return system sizes in \( \text{SYS} \),
  initial state in \( \text{X0} \), state ordering strings
  in \( \text{STR} \), and sample times in \( \text{TS} \).
| 1 \( \text{DX} \) | Return continuous state derivatives in \( \text{SYS} \).
| 2 \( \text{DS} \) | Update discrete states \( \text{SYS} = \text{X}(:,n+1) \)
| 3 \( \text{Y} \) | Return outputs in \( \text{SYS} \).
| 4 \( \text{TNEXT} \) | Return next time hit for variable step sample
  time in \( \text{SYS} \).
| 5 | Reserved for future (root finding).
| 9 \( \text{[]} \) | Termination, perform any cleanup \( \text{SYS}=[] \).

The state vectors, \( X \) and \( X0 \) consists of continuous states followed
by discrete states.

Optional parameters, \( P1, \ldots, Pn \) can be provided to the S-function and
used during any FLAG operation.

When SFUNC is called with FLAG = 0, the following information
should be returned:

\( \text{SYS}(1) \) = Number of continuous states.
\( \text{SYS}(2) \) = Number of discrete states.
\( \text{SYS}(3) \) = Number of outputs.
\( \text{SYS}(4) \) = Number of inputs.

Any of the first four elements in \( \text{SYS} \) can be specified
as -1 indicating that they are dynamically sized. The
actual length for all other flags will be equal to the
length of the input, \( U \).

\( \text{SYS}(5) \) = Reserved for root finding. Must be zero.
\( \text{SYS}(6) \) = Direct feedthrough flag (1=yes, 0=no). The s-function
  has direct feedthrough if \( U \) is used during the FLAG=3
  call. Setting this to 0 is akin to making a promise that
  \( U \) will not be used during FLAG=3. If you break the promise
  then unpredictable results will occur.
\( \text{SYS}(7) \) = Number of sample times. This is the number of rows in \( \text{TS} \).

\( \text{X0} \) = Initial state conditions or \( \text{[]} \) if no states.
\( \text{STR} \) = State ordering strings which is generally specified as \( \text{[]} \).
TS = An m-by-2 matrix containing the sample time (period, offset) information. Where m = number of sample
times. The ordering of the sample times must be:

\[
\begin{align*}
TS &= [0 \ 0], \quad \text{Continuous sample time.} \\
0 \ 1, \quad \text{Continuous, but fixed in minor step} \\
\text{sample time.} \\
\text{PERIOD OFFSET,} \quad \text{Discrete sample time where} \\
\text{PERIOD > 0 & OFFSET < PERIOD.} \\
-2 \ 0]; \quad \text{Variable step discrete sample time} \\
\text{were FLAG=4 is used to get time of} \\
\text{next hit.}
\end{align*}
\]

There can be more than one sample time providing they are ordered such that they are monotonically
increasing. Only the needed sample times should be
specified in TS. When specifying than one
sample time, you must check for sample hits explicitly by
seeing if abs(round(((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
is within a specified tolerance, generally 1e-8. This
tolerance is dependent upon your model's sampling times
and simulation time.

You can also specify that the sample time of the S-function
is inherited from the driving block. For functions which
change during minor steps, this is done by
specifying SYS(7) = 1 and TS = [-1 0]. For functions which
are held during minor steps, this is done by specifying
SYS(7) = 1 and TS = [-1 0].

Copyright (c) 1990-97 by The MathWorks, Inc.
$Revision: 1.9 $

The following outlines the general structure of an S-function.

parameters=[a1 a2 a3 b1 b2]';

switch flag,

\[\text{Initialization}\]

case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

\[\text{Update}\]

case 1,
    sys = mdlDerivatives(t,x,u); \ Update continuous states

\[\text{Outputs}\]

case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9},
    sys = [1]; % do nothing

% Unexpected flags %
% otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

% end sfunmp1

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
% function.
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% function [sys,x0,str,ts]=mdlInitializeSizes
% call simsizes for a sizes structure, fill it in and convert it to
% a
% sizes array.
% Note that in this example, the values are hard coded. This is not
% a
% recommended practice as the characteristics of the block are
% typically defined by the S-function parameters.
% sizes
sizes = simsizes;

sizes.NumContStates = 5; % a1,a2,a3,b1,b2
sizes.NumDiscStates = 0;
sizes.NumInputs = 7; % a1,a2,a3,b1,b2,ModelX,ModelY
sizes.NumOutputs = 16; % [x,xdot,y,ydot],Regressors (12)
sizes.DirFeedthrough = 2;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

% initialize the initial conditions

x0 = [.1,.1,.6,.4,.0.1];
% str is always an empty matrix
str = [];

% initialize the array of sample times
ts = [0 0];

; end mdlInitializeSizes

;
;-----------------------------------------------
; mdlUpdate
; Handle discrete state updates, sample time hits, and major time step
; requirements.
;-----------------------------------------------
; function sys = mdlDerivatives(t,x,u)

if(t>30)
    Gain= diag([.1,1000,100,.1,.1]);
    for i = 1:4,
        Process_data(i)=u(i);
    end
    for i = 1:6,
        v(i)=u(i+4);
        z(i)=u(i+10);
    end
    ModelX=v*diag([1,10000,1000,1,1,1])*[x;1];  %x is the parameters
to be estimated[1,10000,1000,1,1,1];
    if abs(ModelX) > 1.5
        LX =1;
    else
        LX=0;
    end

    ModelY=z*diag([1,10000,1000,1,1,1])*[x;1];
    if abs(ModelY) > 1.5
        LY =1;
    else
        LY=0;
    end

    XPred_error=Process_data(1)-ModelX;
    YPred_error=Process_data(3)-ModelY;
    RegressorsX=v(1:5);
    RegressorsY=z(1:5);
    adot=(XPred_error*RegressorsX+YPred_error*RegressorsY)*Gain;
    %adot(1)=0;
    %adot(2)=0;
    %adot(3)=0;
    %adot(4)=0;
    %adot(5)=0;
    sys = adot;
else
    syrs=zeros(5,1);
end

; end mdlUpdate
% mdlOutputs
% Return the block outputs.

function sys=mdlOutputs(t,x,u)

for i = 1:6,
    v(i)=u(i+4);
    z(i)=u(i+10);
end

ModelX=v*diag([1,10000,1000,1,1,1])*[x;1];  % x is the parameters to be estimated
ModelY=z*diag([1,10000,1000,1,1,1])*[x;1];
if abs(ModelX) > 1.5
    LX =1;
else
    LX=0;
end

sys = [x;ModelX;ModelY];
save ModelX ModelX;
save ModelY ModelY;
end mdlOutputs

[C1,L1]=wavedec(ModelX,3,'db3');  % Perform a level 3 decomposition
[C3,L3]=wavedec(ModelY,3,'db3');  % Perform a level 3 decomposition

C1_A3=appcoef(C1,L1,'db3',3);  % Extract level 3 approximation coefficients
C3_A3=appcoef(C3,L3,'db3',3);  % Extract level 3 approximation coefficients

Model_X =wrcoef('a',C1,L1,'db3',3);  % reconstruct level 3 approximation
Model_Y=wrcoef('a',C3,L3,'db3',3);  % reconstruct level 3 approximation

sys = [x;Model_X;Model_Y];

[C1,L1] = wavedec(ModelX,6,'sym8');
% threshold the decomposition structure [C1,L1].
Model_X = wden(C1,L1,'minimaxi','s','sln',6,'sym8');

[C2,L2] = wavedec(ModelY,6,'sym8');
% threshold the decomposition structure [C2,L2].
Model_Y = wden(C2,L2,'minimaxi','s','sln',6,'sym8');

plot(ModelX,ModelY,'-'), title('Denoising "State Trajectory"')
sys = [x;Model_Y;Model_Y];
 mdlGetTimeOfNextVarHit
\* Return the time of the next hit for this block. Note that the
\* absolute time. Note that this function is only used when you
\* specify a
\* variable discrete-time sample time {\{-2 0\} in the sample time array
\* mdlInitializeSizes.
\*--------------------------------------------------------------------------------
\* function sys=mdlGetTimeOfNextVarHit(t,x,u)
\* sampleTime = 1; \* Example, set the next hit to be one second
\* later.
\* sys = t - sampleTime;
\* end mdlGetTimeOfNextVarHit

mdlTerminate
\* Perform any end of simulation tasks.
\*--------------------------------------------------------------------------------
\* function sys=mdlTerminate(t,x,u)
\* sys = [];
\* end mdlTerminate

plot('Parameters(t,3)')
plot('Parameters(t,1:5)')

function [sys,x0,str,ts] = Rotor_Test(t,x,u,flag,parameters)
\*SFUNCMPL General M-file S-function template
\*With M-file S-functions, you can define your own ordinary
\*differential
\*equations (ODEs), discrete system equations, and/or just about
\*any type of algorithm to be used within a Simulink block diagram.
\*The general form of an M-File S-function syntax is:
\*[SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,P1,...,Pn)
\*What is returned by SFUNC at a given point in time, T, depends on
\*the value of the FLAG, the current state vector, X, and the current
\*input vector, U.
\*FLAG RESULT DESCRIPTION
\*--------------------------------------------------------------------------------
\*0 [SIZES,X0,STR,TS] Initialization, return system sizes in SYS,
\*in STR, and sample times in TS.
\*1 DX Return continuous state derivatives in SYS.
\*2 DS Update discrete states SYS = X(n+1)
\*3 Y Return outputs in SYS.
TNEXT  Return next time hit for variable step sample
time in SYS.
5  Reserved for future (root finding).
2  []  Termination, perform any cleanup SYS=[].

The state vectors, X and X0 consists of continuous states followed
by discrete states.

Optional parameters, F1,...,Fn can be provided to the S-function
and used during any FLAG operation.

When SFUNC is called with FLAG = 0, the following information
should be returned:

SYS(1) = Number of continuous states.
SYS(2) = Number of discrete states.
SYS(3) = Number of outputs.
SYS(4) = Number of inputs.
Any of the first four elements in SYS can be specified
as -1 indicating that they are dynamically sized. The
actual length for all other flags will be equal to the
length of the input, U.
SYS(5) = Reserved for root finding. Must be zero.
SYS(6) = Direct feedthrough flag (1=yea, 0=no). The s-function
has direct feedthrough if U is used during the FLAG=3
call. Setting this to 0 is akin to making a promise that
U will not be used during FLAG=3. If you break the promise
then unpredictable results will occur.
SYS(7) = Number of sample times. This is the number of rows in TS.

X0  = Initial state conditions or [] if no states.
STR= State ordering strings which is generally specified as [;].

TS  = An m-by-1 matrix containing the sample time
      (period, offset) information. Where m = number of sample
times. The ordering of the sample times must be:

TS = [] 0, 0;  Continuous sample time.
    1, 1;  Continuous, but fixed in minor step
           sample time.
    PERIOD OFFSET;  Discrete sample time where
                      PERIOD > 0 & OFFSET < PERIOD.
                      -2 0;  Variable step discrete sample time
                      were FLAG=4 is used to get time of next hit.

There can be more than one sample time providing
they are ordered such that they are monotonically
increasing. Only the needed sample times should be
specified in TS. When specifying than one
sample time, you must check for sample hits explicitly by seeing if
abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
is within a specified tolerance, generally 1e-6. This
tolerance is dependent upon your model's sampling times
and simulation time.

You can also specify that the sample time of the S-function is inherited from the driving block. For functions which change during minor steps, this is done by specifying SYS(7) = 1 and TS = [-1 0]. For functions which are held during minor steps, this is done by specifying SYS(7) = 1 and TS = [-1 0].

Copyright (c) 1990-96 by The MathWorks, Inc.
$Revision: 1.9 $

The following outlines the general structure of an S-function.

parameters=[al a2 a3 b1 b2]';

switch flag,

    case 0,
        [sys,x0,str,ts]=mdlInitializeSizes;

    case 2,
        sys=mdlUpdate(t,x,u,parameters);

    case 3,
        sys=mdlOutputs(t,x,u);

    case {1,4,9}
        sys = [] ; % do nothing

    otherwise
        error(['Unhandled flag = ',num2str(flag)]);

end

end sfuntmpl
function [sys,x0,str,ts]=mdlInitializeSizes
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
% Note that in this example, the values are hard coded. This is not a
% recommended practice as the characteristics of the block are typically
defined by the S-function parameters.
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 4;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes(sizes);
%
% initialize the initial conditions
%
x0 = ones(4,1); % rand(4,1);
%
% str is always an empty matrix
% str = [];
%
% initialize the array of sample times
% ts = [1 0];
% end mdlInitializeSizes

% mdIUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
function sys=mdlUpdate(t,x,u,parameters)

T=.005;
M=1;
\[ K_s = 1000000.0; \]
\[ CSU_\omega_s = \sqrt{K_s/M}; \]
\[ CSU_\omega = 1; \]

diam = .03; \quad \text{Disk Diameter}
clrnc = 0.00072; \quad \text{Clearance}
rho = \frac{\text{diam}}{2 \cdot \text{clrnc}}; \quad \text{Normalized Disk Radius}
NoiseLevel = 0.0;

if \( x(1) < 1 \cdot 12 \) \& \( x(3) < 1.0 \cdot 12 \)
\[
\theta = 0;
\]
else
\[
\text{if} \quad x(1) < 1 \cdot 12 \quad \&\quad x(3) > 1 \cdot 12
\]
\[
\theta = \frac{\pi}{2};
\]
else
\[
\theta = \arctan(x(3)/x(1));
\]
end
\[
R = \sqrt{x(1) \cdot x(1) + x(3) \cdot x(3)};
\]
\[
\Delta = R - 1;
\]

if \( R > 0 \)
\[
\phi = x(1) \cdot x(4) - x(3) \cdot x(2); \quad \text{in the code results match those in Figure 2.5}
\]
\[
\phi = \frac{\phi}{R \cdot R \cdot CSU_\omega_s}; \quad \text{phi} = \phi \cdot R + CSU_\omega_s \cdot \rho;
\]
\[
\text{Sign} \phi = \text{sign}(\phi);
\]
else
\[
\text{Sign} \phi = 0
\]
end
\[
\phi_c = (\Delta^2.5) \cdot x(1)/R;
\]
\[
\phi_s = (\Delta^2.5) \cdot x(3)/R;
\]
\[
x_1 = x(1) + T \cdot x(2);
\]
\[
x_2 = x(2) + T \cdot \left(-x(1) \cdot 2 \cdot \text{parameters}(1) \cdot x(2) + \text{parameters}(4) \cdot \sqrt{1-u^2}\right);
\]
if \( \Delta > 0 \) \& \( R > 0 \)
\[
x_2 = x_2 + T \cdot \left(-\text{parameters}(2) \cdot \phi_c + \text{parameters}(3) \cdot \text{Sign} \phi \cdot \phi_s\right);
\]
end
\[
x_3 = x(3) + T \cdot x(4);
\]
\[
x_4 = x(4) + T \cdot \left(-x(3) \cdot 2 \cdot \text{parameters}(1) \cdot x(4) + \text{parameters}(4) \cdot \left(1-u^2\right)\right);
\]
if \( \Delta > 0 \) \& \( R > 0 \)
\[
x_4 = x_4 + T \cdot \left(-\text{parameters}(2) \cdot \phi_s - \text{parameters}(3) \cdot \text{Sign} \phi \cdot \phi_c\right);
\]
end
\[
sys = [x_1 \ x_2 \ x_3 \ x_4];
\]
end mdUpdate

\[ \text{mdlOutputs} \]
\[ \text{Return the block outputs.} \]
function sys=mdlOutputs(t,x,u)

sys = x;

; end mdlOutputs
;

function sys=mdlGetTimeOfNextVarHit(t,x,u)

; Return the time of the next hit for this block. Note that the result is absolute time. Note that this function is only used when you specify a variable discrete-time sample time [0 0] in the sample time array in mdlInitializeSizes.

function sys=mdlGetTimeOfNextVarHit(t,x,u)

;sampleTime = 1;   \ Example, set the next hit to be one second later.
;sys = t - sampleTime;

; end mdlGetTimeOfNextVarHit
;

function sys=mdlTerminate(t,x,u)

;sys = [];
; end mdlTerminate

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [sys,x0,str,ts] = Rotor_Regressors_c(t,x,u,flag)

%SFUNTMP General M-file S-function template
%With M-file S-functions, you can define your own ordinary
%differential equations (ODEs), discrete system equations, and/or just about
%any type of algorithm to be used within a Simulink block diagram.

%The general form of an M-file S-function syntax is:
% [SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,F1,...,Fn)

%What is returned by SFUNC at a given point in time,T, depends on the
%value of the FLAG, the current state vector, X, and the current
%input vector, U.

%FLAG    RESULT    DESCRIPTION
%----------------------------------
%  0     [SIZES,X0,STR,TS]  Initialization, return system sizes in SYS,
%                        initial state in X0, state ordering strings
%                        in STR, and sample times in TS.
%  1     DX            Return continuous state derivatives in SYS.
%  2     DS            Update discrete states SYS = X(n-1)
%  3     Y             Return outputs in SYS.
%  4     TNEXT         Return next time hit for variable step sample
%                        time in SYS.
%  5     Reserved for future (root finding).
%  6     []            Termination, perform any cleanup SYS=[].

The state vectors, X and X0 consists of continuous states followed
by discrete states.

Optional parameters, P1,...,Pn can be provided to the S-function and
used during any FLAG operation.

When SFUNC is called with FLAG = 0, the following information
should be returned:

SYS(1) = Number of continuous states.
SYS(2) = Number of discrete states.
SYS(3) = Number of outputs.
SYS(4) = Number of inputs.
Any of the first four elements in SYS can be specified
as -1 indicating that they are dynamically sized. The
actual length for all other flags will be equal to the
length of the input, U.
SYS(5) = Reserved for root finding. Must be zero.
SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function
has direct feedthrough if U is used during the FLAG=3
call. Setting this to 0 is akin to making a promise that
U will not be used during FLAG=3. If you break the promise
then unpredictable results will occur.
SYS(7) = Number of sample times. This is the number of rows in TS.

X0 = Initial state conditions or [] if no states.

STR = State ordering strings which is generally specified as [],.

TS = An m-by-2 matrix containing the sample time (period, offset) information. Where m = number of sample times. The ordering of the sample times must be:

\[
\begin{align*}
0 & & 0, & & \text{Continuous sample time.} \\
0 & & 1, & & \text{Continuous, but fixed in minor step sample time.} \\
\text{PERIOD} & & \text{OFFSET}, & & \text{Discrete sample time where PERIOD} > 0 & \text{ & OFFSET} < \text{PERIOD.} \\
-2 & & 0; & & \text{Variable step discrete sample time.}
\end{align*}
\]

There can be more than one sample time providing they are ordered such that they are monotonically increasing. Only the needed sample times should be specified in TS. When specifying more than one sample time, you must check for sample hits explicitly by

\[
\text{sim} \left( \text{round} \left( \frac{\text{T-OFFSET}}{\text{PERIOD}} \right) \right) = \frac{\text{T-OFFSET}}{\text{PERIOD}}
\]

This is within a specified tolerance, generally 1e-6.

To tolerance is dependent upon your model's sampling times and simulation time.

You can also specify that the sample time of the S-function is inherited from the driving block. For functions which change during minor steps, this is done by

specifying SYS(7) = 1 and TS = [-1 0]. For functions which are held during minor steps, this is done by specifying SYS(7) = 1 and TS = [-1 0].

Copyright (c) 1990-97 by The MachWorks, Inc.

\$Revision: 1.9 $

The following outlines the general structure of an S-function.

\[
\text{parameters}=[a1 a2 a3 b1 b2]';
\]

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

case 1,
    sys=mdlDerivatives(t,x,u);

case 3,
    sys=mdlOutputs(t,x,u);

case [1,2,4,9]
    sys = [];  % do nothing

otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

end sfunmp1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
% function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sys,x0,str,ts]=mdlInitializeSizes
% Call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded. This is not
% a
% recommended practice as the characteristics of the block are
% typically
% defined by the S-function parameters.
%
sizes = simsizes;

sizes.NumContStates  = 0;  % x,y
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 16;  % x,y, and Regressors Z[1:5] AND V
[1:5]
sizes.NumInputs = 6; \ x,x',y,y', x'',y''
sizes.NumSampleTimes = 1; \ at least one sample time is needed

sys = simsizes(sizes);

% initialize the initial conditions

\x0 =0.01*rand(4,1); \rand(4,1);
x0=[1];
%------------------- Reading x and y-------------------

% str is always an empty matrix
str = [];

% initialize the array of sample times

ts = [0 0];

% end mdlInitializeSizes

%============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%============================================================================
% function sys=mdlDerivatives(t,x,u)

\ i=i-1;
% if i<1024 then
\ x1 = xdof(i);
\ x3 = ydof(i);
% end

\ sys = [x1 x3]';

% end mdlUpdate

%============================================================================
% mdlOutputs
% Return the block outputs.
%============================================================================
% function sys=mdlOutputs(t,x,u)
%Calculate regressors as feed through output signals
M=1;
KS = 100000.0;
CSU_omega_s=sqrt(Ks/M);
CSU_omega=1;

diam=.03; ; Disk Diameter
clrnc =0.00072; ; Clearance
rho = diam/(2*clrnc);  ; Normalized Disc Radius
RubNoise=0.01;
LinearNoiseLevel=0.01;
SensNoise=0.1;

if(u(1)<1e-12 & u(3)<1.e-12)
  theta = 0;
else
  if (u(1)<1e-12 & u(3)>1e-12)
    theta=pi/2;
  else
    theta=atan(u(3)/u(1));
  end

R=sqrt(u(1)*u(1)+u(3)*u(3));
delta= R-1;

if(R>0)
  phi = u(1)*u(4)-u(3)*u(2);
  phi = phi/(R*R*CSU_omega_s);
  psi=phi*R*CSU_omega*rho;
  SignPsi=sign(psi);
else
  SignPsi=0;
end

if(R>0)
  phi_c=(delta^1.5)*u(1)/R;
  phi_s=(delta^1.5)*u(2)/R;
end

v=zeros(6,1); ; CHANGING IT TO 5 THE 6TH REGRESSOR WILL BE CALCULATED OUTSIDE THE S FUNCTION
z=zeros(6,1); ; CHANGING IT TO 5 THE 6TH REGRESSOR WILL BE CALCULATED OUTSIDE THE S FUNCTION

v(1)=-2*u(2);
if (delta>0 & R>0)
  v(2)=-phi_c;
  v(3)=SignPsi*phi_s;
else
  v(2)=0.0;
  v(3)=0.0;
end
v(4)=cos(t);
v(5)=0;
v(6)=u(5);
z(1) = -2*u(4);
if (delta>0 & R>0)
    z(2)=phi_s;
    z(3)=SignPsi*phi_c;
else
    z(2)=0.0;
    z(3)=0.0;
end
z(4)=sin(t);
z(5)=-l;
z(6)=u(6);

sys = [u(1);u(2);u(3);u(4);v;z];

end md1Outputs

function [sys,x0,str,ts] = Rotor_Process_c(t,x,u,flag)
global i
global xdofa
global ydofa

%SFUNTMPL General M-file S-function template
%With M-file S-functions, you can define your own ordinary
differential
%equations (ODEs), discrete system equations, and/or just about
%any type of algorithm to be used within a Simulink block diagram.
%
The general form of an M-File S-function syntax is:
% [SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,F1,...,Fn)
%
%What is returned by SFUNC at a given point in time, T, depends on the
%value of the FLAG, the current state vector, X, and the current
%input vector, U.

% FLAG   RESULT                  DESCRIPTION
% -----   ------                 -------------------------------
%  0       [SIZES,X0,STR,TS]     Initialization, return system sizes in SYS,
%                               initial state in X0, state ordering strings
%                               in STR, and sample times in TS.
%  1       X  Return continuous state derivatives in SYS.
%  2       DS Update discrete states SYS = X(m-1)
%  3       Y  Return outputs in SYS.
%  4       TNEXT Return next time hit for variable step sample
%              time in SYS.
%  5       Reserved for future (root finding).
%  9       []  Termination, perform any cleanup SYS=[].
%
The state vectors, X and X0 consists of continuous states followed
by discrete states.
Optional parameters, P1, ..., Pn can be provided to the S-function and
used during any FLAG operation.

When SFUNC is called with FLAG = 0, the following information
should be returned:

SYS(1) = Number of continuous states.
SYS(2) = Number of discrete states.
SYS(3) = Number of outputs.
SYS(4) = Number of inputs.
Any of the first four elements in SYS can be specified
as -1 indicating that they are dynamically sized. The
actual length for all other flags will be equal to the
length of the input, U.
SYS(5) = Reserved for root finding. Must be zero.
SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function
has direct feedthrough if U is used during the FLAG=3
call. Setting this to 0 is akin to making a promise that
U will not be used during FLAG=3. If you break the promise
then unpredictable results will occur.
SYS(7) = Number of sample times. This is the number of rows in TS.

XO      = Initial state conditions or [] if no states.
STR     = State ordering strings which is generally specified as [].
TS      = An m-by-2 matrix containing the sample time
        information. Where m = number of sample
times. The ordering of the sample times must be:

TS = [0 0, 0, 1,   : Continuous sample time.
      0 1,  : Continuous, but fixed in minor step
      PERIOD OFFSET,  : Discrete sample time where
      PERIOD > 0 & OFFSET < PERIOD.
      -2 0];  : Variable step discrete sample time
      were FLAG=4 is used to get time of
      next hit.

There can be more than one sample time providing
they are ordered such that they are monotonically
increasing. Only the needed sample times should be
specified in TS. When specifying more than one
sample time, you must check for sample hits explicitly by
seeing if
abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
is within a specified tolerance, generally 1e-6. This
tolerance is dependent upon your model's sampling times
and simulation time.

You can also specify that the sample time of the S-function
is inherited from the driving block. For functions which
change during minor steps, this is done by
specifying SYS(7) = 1 and TS = [-1 0]. For functions which
are held during minor steps, this is done by specifying
SYS(7) = 1 and TS = [-1 0].

Copyright (c) 1990-97 by The MathWorks, Inc.
$Revision: 1.9 S$

The following outlines the general structure of an S-function.
parameters=[a1 a2 a3 b1 b2]';

switch flag,

%*****************************************************************************
% Initialization 
%*****************************************************************************

case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

%*****************************************************************************
% Update 
%*****************************************************************************

case 1,
    sys=mdlDerivatives(t,x,u);

%*****************************************************************************
% Outputs 
%*****************************************************************************

case 3,
    sys=mdlOutputs(t,x,u);

case {1,2,4,9}
    sys = []; % do nothing

%*****************************************************************************
% Unexpected flags 
%*****************************************************************************

otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end;

% and sfuntmpl

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
% function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sys,x0,str,ts]=mdlInitializeSizes
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
% Note that in this example, the values are hard coded. This is not a recommended practice as the characteristics of the block are typically defined by the S-function parameters.

global i
global xdofa
global ydofa

sizes = simsizes;

sizes.NumContStates = 0; % x,y
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2; % x,y, and Regressors E[1:3] AND V [1:5]
sizes.NumInputs = 0; % a1,a2,a3,b1,b2 ************* Delete ALL THE INPUT
sizes.DirFeedthrough = 2; % Regressors for parameter estimator
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

% initialize the initial conditions

x0=[];
x0 = 0.51*rand(4,1); % rand(4,1);

% str is always an empty matrix
str = [];

% initialize the array of sample times
ts = [0 0];

% --------------------- Reading x and y ---------------------
clc c;
i=0;
xdofa=zeros(1024,1);
ydofa=zeros(1024,1);
xdofa=dlmread('xdofa.txt');
ydofa=dlmread('ydofa.txt');

% mdlinitsizes

% mdldervatives
% Handle discrete state updates, sample time hits, and major time step
function sys=mdlDerivatives(t,x,u)
global i
global xdofa
global ydofa
i=i+1;
w1 = xdofa(i);
w2 = ydofa(i);
sys = [w1;w2];
end mdlDerivatives

mdlOutputs
Return the block outputs.

function sys=mdlOutputs(t,x,u)
global i
global xdofa
global ydofa
i=i+1;
w1 = xdofa(i);
w2 = ydofa(i);
sys = [w1;w2];
end mdlOutputs

function [sys,x0,str,ts] = Rotor_Estimator_c(t,x,u,flag)
SFCNTMPL General M-file S-function template
With M-file S-functions, you can define your own ordinary differential
equations (ODEs), discrete system equations, and/or just about
any type of algorithm to be used within a Simulink block
diagram.

The general form of an M-File S-function syntax is:
[SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,F1,...,Fn)

What is returned by SFUNC at a given point in time, T, depends
on the value of the FLAG, the current state vector, X, and the current
input vector, U.

<table>
<thead>
<tr>
<th>FLAG</th>
<th>RESULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[SIZES,X0,STR,TS]</td>
<td>Initialization, return system sizes in SYS, initial state in X0, state ordering strings in STR, and sample times in TS.</td>
</tr>
<tr>
<td>1</td>
<td>DX</td>
<td>Return continuous state derivatives in SYS.</td>
</tr>
<tr>
<td>2</td>
<td>DS</td>
<td>Update discrete states SYS = X(n+1)</td>
</tr>
</tbody>
</table>
Return outputs in SYS.
Return next time hit for variable step sample
time in SYS.
Reserved for future (root finding).
[] Termination, perform any cleanup SYS=[].

The state vectors, X and X0 consists of continuous states followed by discrete states.
Optional parameters, PL,...,Pn can be provided to the S-function and used during any FLAG operation.
When SFUNC is called with FLAG = 0, the following information should be returned:

SYS(1) = Number of continuous states.
SYS(2) = Number of discrete states.
SYS(3) = Number of outputs.
SYS(4) = Number of inputs.
Any of the first four elements in SYS can be specified as -1 indicating that they are dynamically sized. The actual length for all other flags will be equal to the length of the input, U.
SYS(5) = Reserved for root finding. Must be zero.
SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function has direct feedthrough if U is used during the FLAG=3 call. Setting this to 0 is akin to making a promise that U will not be used during FLAG=3. If you break the promise then unpredictable results will occur.
SYS(7) = Number of sample times. This is the number of rows in TS.

X0    = Initial state conditions or [] if no states.
STR   = State ordering strings which is generally specified as [].
TS    = An m-by-2 matrix containing the sample time (period, offset) information. Where m = number of sample times. The ordering of the sample times must be:

TS = [0 0], : Continuous sample time.
0 1, : Continuous, but fixed in minor step sample time.
PERIOD OFFSET, : Discrete sample time where PERIOD > 0 & OFFSET < PERIOD.
-2 0]; : Variable step discrete sample time were FLAG=4 is used to get time of next hit.

There can be more than one sample time providing they are ordered such that they are monotonically increasing. Only the needed sample times should be specified in TS. When specifying than one
sample time, you must check for sample hits explicitly by
seeing if
abs(round Educación (T-OFFSET)/PERIOD) = (T-OFFSET)/PERIOD
is within a specified tolerance, generally 1e-6. This
tolerance is dependent upon your model's sampling times
and simulation time.

You can also specify that the sample time of the S-function
is inherited from the driving block. For functions which
change during minor steps, this is done by
specifying SYS(7) = 1 and TS = [-1 0]. For functions which
are held during minor steps, this is done by specifying
SYS(7) = 1 and TS = [-1 0].

The following outlines the general structure of an S-function.
parameters=[a1 a2 a3 b1 b2];

switch flag,
	\mbox{Initialization,}\nonumber
case 0,\nonumber
\mbox{[sys,x0,str,ts]=mdlInitializeSizes;}
\mbox{Update:\nonumber}
case 2,\nonumber
\mbox{sys=mdlUpdate(t,x,u); \ Update discrete states}\nonumber
\mbox{Outputs:\nonumber}
case 3,\nonumber
\mbox{sys=mdlOutputs(t,x,u);}\nonumber
\mbox{case \{1,4,9\},}\nonumber
\mbox{sys = []; \ do nothing}\nonumber
\mbox{Unexpected flags:\nonumber}
\mbox{otherwise}\nonumber
\mbox{error([''Unhandled flag = '',num2str(flag)]);}
end\nonumber
\mbox{end sfuntmp1}\nonumber
\mbox{mdlInitializeSizes}\nonumber
\mbox{Return the sizes, initial conditions, and sample times for the S-}
f
function [sys, x0, str, ts] = mdlInitializeSizes
    \ call simsize for a sizes structure, fill it in and convert it to
    \ a sizes array.
    \ Note that in this example, the values are hard coded. This is not
    \ recommended practice as the characteristics of the block are 
    \ typically defined by the S-function parameters.
    sizes = simsizes;

    sizes.NumContStates = 0; \ a1, a2, a3, b1, b2
    sizes.NumDiscStates = 5;
    sizes.NumOutputs = 7; \ a1, a2, a3, b1, b2, ModelX, ModelY
    sizes.NumInputs = 16; \ (x, xdot, y, ydot, Regressors 1:2)
    sizes.DirFeedthrough = 2;
    sizes.NumSampleTimes = 1; \ at least one sample time is needed

    sys = simsizes(sizes);

    \ initialize the initial conditions

    x0 = [.1, 0, 0, .4, 0.1]';

    \ str is always an empty matrix
    str = [];

    \ initialize the array of sample times
    ts = [.05 0];

    \ end mdlInitializeSizes

    \=====================================================================
    \ mdlUpdate
    \ Handle discrete state updates, sample time hits, and major time
    \ step requirements.
    \=====================================================================
    function sys = mdlUpdate(t, x, u)
    \ t>30
    Gain = diag([.1, 1000, 100, .1, .1]);
    for i = 1:5,
        \ Process_data(i) = u(i);
    end
    for i = 1:6,
\begin{verbatim}
v(i)=u(i+4);
z(i)=u(i+10);
end
ModelX=v*diag([[1,10000,1000,1,1,1]])*[x;1];  \text{x is the parameters to be estimated}
ModelY=z*diag([[1,10000,1000,1,1,1]])*[x;1];

XPred_error=Process_data(1)-ModelX;
YPred_error=Process_data(3)-ModelY;
RegressorX=v(1:5);
RegressorY=z(1:5);
adot=(XPred_error*RegressorX+YPred_error*RegressorY)*Gain;
\text{adot}(1)=0;
\text{adot}(2)=0;
\text{adot}(3)=0;
\text{adot}(4)=0;
\text{adot}(5)=0;

sys = adot; + (Gain*Pred_error*Regressor(1),0,0,0,0);  \text{u is the regressor vector (EPSI)}
else
\text{sys}=zeros(5,1);
end

end mdlUpdate

%===============================================================================
% mdlOutputs
% Return the block outputs.
%===============================================================================

function sys=mdlOutputs(t,x,u)

for i = 1:6,
v(i)=u(i+4);
z(i)=u(i+10);
end
ModelX=v*diag([[1,10000,1000,1,1,1]])*[x;1];  \text{x is the parameters to be estimated}
ModelY=z*diag([[1,10000,1000,1,1,1]])*[x;1];
if abs(ModelX) > 1.5
LX =1;
else
LX=0;
end

sys = [x;ModelX;ModelY];
\text{save ModelX ModelX;}
\text{save ModelY ModelY;}
end mdlOutputs
%===============================================================================

[C1,L1]=wavedec(ModelX,3,'db3');  \text{Perform a level 3 decomposition}
[C3,L3]=wavedec(ModelY,3,'db3');  \text{Perform a level 3 decomposition}
\end{verbatim}
C1_A3=appcoef(C1,L1,'db3',3); % Extract level 3 approximation coefficients
C3_A3=appcoef(C3,L3,'db3',3); % Extract level 3 approximation coefficients

Model_X = wrcoef('a',C1,L1,'db3',3); % reconstruct level 3 approximation
Model_Y = wrcoef('a',C3,L3,'db3',3); % reconstruct level 3 approximation

sys = [x;Model_X;Model_Y];

%.tplw = wavelet('l1',6,'sym6'); % threshold the decomposition structure [C1,L1]
Model_X = wdencmp('l1',C1,L1,'sym6',6,'sym6');

%tplw = wavelet('l2',6,'sym6'); % threshold the decomposition structure [C2,L2]
Model_Y = wdencmp('l2',C2,L2,'sym6',6,'sym6');

plot(Model_X,Model_Y,'.-', title('Denoising State Trajectory'));

sys = [x;Model_X;Model_Y];

% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time. Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array 
in
% mdlInitializeSizes.
%function sys=mdlGetTimeOfNextVarHit(t,x,u)

%sampleTime = 1; % Example, set the next hit to be one second later.
%sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

% mdlTerminate
% Perform any end of simulation tasks.
%function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate