Limiting Deviation Method for Coupling Codes with Adaptive Window-Size Control using Digital Filters

by

Rohan Palaniappan

A research paper submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Limiting Deviation Method for Coupling Codes with Adaptive Window-Size Control using Digital Filters

Rohan Palaniappan
Master of Science
Graduate Department of Computer Science
University of Toronto
2015

Multi-physics systems model multiple simultaneous physical phenomena that affect one another. Simulating such systems can be done using the Operator Splitting (OS) method, wherein separate simulations are run concurrently for each phenomenon and data is exchanged between the simulations periodically. In this work, we explore, in the context of the OS method, what we call the Limiting Deviation (LD) and Limiting Deviation with Interior Check (LDIC) methods, which indirectly limit the global solution error in an unconventional way. We test the LD and LDIC methods using "canned data" for five test cases using thirteen different adaptive step-size controllers. We identify the best controllers for two experiments, point out undesired behaviour in certain controllers and examine the effect of two key parameters. Finally, we discuss implementation options. While our results are quite positive, we consider them preliminary, because the tests do not factor-in the coupling between the separate simulations.

Contents

Chapter 1

Introduction

1.1 The Code-Coupling Problem

A mathematical model of a system, be it physical, financial, etc., allows one to predict the approximate state of the system at some future time t, given the initial conditions at $t_0 < t$ and the boundary conditions, if appropriate, assuming that the model is valid for the time period $[t_0, t]$. For example, Newton's law of universal gravitation allows one to predict approximate values for the position and velocity of a planet that is in orbit around the Sun at some future time t, given the position and velocity of the planet at some time $t_0 < t$ (there are no boundary conditions here). The solution of the mathematical equations associated with a model of the state of a system over a period of time is referred to as a simulation. Simulations are useful for a variety of reasons: for example, determining the safety compliance of an engineering system, such as a nuclear reactor, or determining the compliance of an engineering system to the functional and performance requirements, such as for an aeroplane.

Sometimes, the system that requires simulation consists of distinct components. For example, in certain physical systems, known as multi-physics systems, there are multiple simultaneous physical phenomena that affect one another. An example of a multi-physics system is thermo-mechanical coupling, where there are heat transfer and structural deformation phenomena acting at the same time in a coupled way [1]. Another example is electric field-structural coupling, a.k.a. a *Piezoelectric* system, where there are electric field and mechanical deformation phenomena acting at the same time in a coupled way. To be more precise, the coupling in a Piezoelectric system works as follows. Applying an electric field to a Piezoelectric material causes it to deform, which in turn causes an electric field, which in turn causes the material to deform and so on. (A Piezoelectric material is a type of material that exhibits this behaviour.) [1]. For additional examples of multi-physics systems, see [1]. We refer to systems that consist of distinct components, such as multi-physics systems, as *multi-component systems* (this is a general term which covers different types of systems, such as financial, biological, physical, etc.).

How are multi-component systems simulated? One approach is to run separate simulations for each component concurrently, exchanging data between the simulations periodically. This is referred to as the operator-splitting (OS) method [7]. Another possibility is to use a mono-block approach wherein

the entire system is simulated as one large system. This document is concerned with the first of these two approaches. (There are some drawbacks to the OS method but they can be mitigated using certain remedies suggested in [7], which are briefly discussed in §1.2, the Literature Review section). For the OS method, the sizes of the intervals between the data exchanges are adaptively modified so as to minimize the number of data exchanges while at the same time meeting the accuracy requirements for the global result. In this document, the separate programs used for the simulations are referred to as *codes* and the periods of time between data transfers are referred to as *windows*. The data exchanged between the codes are the values of *state variables*, which are quantities that describe the state of a dynamic system. Examples of state variables are velocity and temperature. The term *variable* is used interchangeably with state variable going forward.

This work was done, in part, to explore the viability of a particular approach to solve a problem that Atomic Energy of Canada Limited (AECL) is looking to address. AECL, a Canadian crown corporation that has been active for more than 60 years, does nuclear science and technology research and development. They perform nuclear reactor simulations and have been doing so for several decades now. Nuclear reactors are a classic example of a multi-physics system: inside a nuclear reactor are thermodynamics and neutronics phenomena acting simultaneously in a coupled way.

For reasons that are not pertinent to this paper, AECL currently simulates its reactors using a method based on *waveform relaxation* [3]. Their method works as follows. The simulation is run several times *to completion* and, for all the runs, the codes do not exchange data at all during the run. For the first simulation run, for the variables that each code needs from the other codes, it uses the initial values of those variables as constants for the entire simulation time period. For the second run, for the variables that each code needs, it uses the values determined in the first run by the other codes. In the third run, it uses the values determined in the *second* run by the other codes, and so on. This is referred to as an *iterative* approach: each simulation run is an iteration. The solution after the final iteration is the final solution. The greater the number of iterations, the greater the accuracy of the final solution.

AECL intends to move from the waveform relaxation approach to an OS approach. In an OS approach, a mechanism for achieving the desired accuracy in the global solution is required. AECL is interested in using a particular unconventional mechanism. To understand it, one first needs to understand a few prerequisite concepts, which are explained next. With many numerical methods, the solution to a problem is discrete, i.e., solution values are obtained at discrete points in the problem domain. For problems in the time domain, the spacing between the discrete time-points is referred to as *time-steps*. Next, with many numerical methods, the solution obtained by the method should meet some prescribed accuracy constraint. That is, the difference between the discrete solution obtained by the numerical method and the true solution at the time-points of the discrete solution should be less than some prescribed tolerance for all the time-points of the discrete solution. These differences are referred to as the *global error* [4]. With many numerical methods, in order to achieve the prescribed global error tolerance, the method limits a *local error* estimate at every time-step. Local error for a time-step is the difference between the exact solution at the end of the time-step and the numerical solution at the end of the time-step, given that the initial values at the beginning of the time-step are the same in both cases. Local error estimates can be obtained in many ways. A commonly used technique in numerically solving ordinary differential equations (ODEs) is the one-step-two-half-steps technique, which works as follows: we compute the solution values at the time-step endpoint; then, we redo the time-step, now in two steps

of equal size; finally, we compute the difference in the values obtained using the full step and the values obtained using the two half-steps. This difference can be used to estimate the local error at the time-step endpoint. As mentioned before, controlling the local error indirectly controls the global error. For the relationship between the local and global errors and an introduction to numerically solving ODEs, refer to [4].

Just as with ODEs, we want to limit the global error of the solution of a multi-component system so as to meet the prescribed accuracy tolerance by limiting a local error estimate over each window. That is what is typically done in the OS method. AECL wants to explore the viability of using the OS coupling method with an alternate way of limiting the global error: instead of limiting estimates of the local errors of the state variables, the deviations of the values of the state variables from the start to the end of each window is limited. We call this approach the *limiting deviation (LD) method*. Note that this method has several significant disadvantages. They are stated briefly here and explained in full later in the paper. First, it is not as efficient in terms of the amount of data transferred and amount of computation required. This is because deviations don't necessarily imply local error. Second, the tolerances for the deviations that will limit the local errors to the desired values will have to be determined by running the simulation multiple times and knowing the true solution (or having a very good approximation of it). Third, the tolerances required to limit the local errors to the same desired values in a different simulation may be different! So, one may have to redo the simulation runs to determine the correct deviation tolerances for the new simulation. Despite these potential disadvantages, we have found that this approach does work well for the simple test problems that we consider in this paper.

We test the LD method and an extension, the *limiting deviation with internal checks* (*LDIC*) method, using *canned data*, which is AECL's preferred approach. By "canned data", we mean that the values of all the variables of all the codes are predetermined and so, the values exchanged between the codes are also predetermined. As such, the codes do not actually affect one another. Note that this work applies to multi-component systems in general, not just multi-physics systems or just nuclear reactors.

Our results for these tests are quite positive. However, we consider these results preliminary in that they do not take into account the effects of coupling between the codes. Therefore, we strongly recommend that further testing of the LD and LDIC methods be performed on test problems that exhibit coupling between the variables in the codes.

Deviation can be measured using several different extrapolation methods. We consider two in this document: *constant extrapolation* (CE) and *linear extrapolation* (LE), which are explained in more detail in Chapter 2. We test the LD and LDIC methods using many different adaptive time-stepping controllers for five test cases for both CE and LE. In particular, we determine if one or more of the controllers provide better "performance" than the rest for the five test cases. By better performance, we mean smaller overall computational work given that the controller only accepts windows for which the deviation is less than or equal to the tolerance. As explained in Chapter 4, the larger (smaller) the amount of overall computational work, the longer (shorter) the overall computing time. As is also explained in Chapter 4, given certain reasonable assumptions, the difference in performance between two controllers is determined primarily by the difference in their number of rejected windows (assuming that the codes do not store the values computed for the most recent window in memory, as we explain in Chapter 4; if they did, the difference in performance between two controllers is equally impacted by

the difference in the number of accepted windows and the difference in the number of rejected windows; we were not aware of this alternate implementation until late in our work and so, we assume the codes do not store the values in memory).

1.2 Literature Overview

The OS technique is the traditional approach for simulating reactors in the nuclear industry [7]. The reason for this is that, historically, the various components of the nuclear reactor multi-physics problem have been simulated separately and, so, employing the OS technique allows for existing legacy codes to be reused. That is beneficial because adopting the mono-block approach instead would require developing a new code and performing extensive code verification and validation work, both of which are expensive, both monetarily and in terms of time [7].

Coupling of the codes with the OS technique can be done in several ways. One approach is a Gauss-Seidel type of coupling wherein, for a given window, one code computes a numerical solution for its subsystem, then passes on some data to another code, which then computes a numerical solution for its subsystem, then passes on its data to a third code and so on, until all codes have executed the window [7]. Another approach is a block Jacobi type of coupling wherein all the codes execute the window at the same time, then exchange data, then execute the next window at the same time, then exchange data, and so on. In both cases, the steps for a window are not iterated. Ragusa and Mahadevan show that regardless of the order of convergence of the methods used by the codes, the OS technique reduces the global order of convergence to first order!

As Ragusa and Mahadevan point out in [7], there exist two "remedies" one can employ to overcome the loss of accuracy inherent in the conventional OS technique. The first is to use "higher degree of linearization" for the "treatment of the lagged nonlinear terms". The second "remedy" is to iterate over each window. That is, after the values of all the coupled variables are computed by all the codes for a given window, the computation is repeated again on the same window. For the second iteration, each code uses the values of the variables which it requires that are computed by the other codes in the first iteration, in the case of the Jacobi iteration, or the most recently computed variables, in the case of Gauss-Seidel. This step is repeated some finite number of times. Then, the next window is executed in the same way and then, the window after that and so on. Note that this is similar to the waveform relaxation technique described in §1.1. Note also that the iterations can be *accelerated*, meaning the rate of convergence of the solution to the "true" solution can be increased; the "true" solution here is the exact solution for the same problem with the following difference: the initial time point is the window starting point and the initial conditions are the window starting point's values (which are equal to the previous window's endpoint values).

To test numerical methods for the nuclear reactor simulation application, many studies have used *simplified models* of nuclear reactors, which are usually systems of PDEs or ODEs [6], [7]. For example, in [7], Ragusa and Mahadevan use the well-known PRKE model and a 1-D model to do their testing of the OS technique with the above-mentioned remedies. Housiadas uses a lumped-parameters 1-D model in [6] to simulate a small research nuclear reactor. On the other hand, sophisticated codes are used by some studies too, for example in [10].

There has been much work in the ODE area over the years on time-step controllers. In 1988, Gustafsson et al. pioneered the use of control theory to analyse and develop step size controllers [5]. This proved to be successful and valuable. As a result, more work has been done over the years in applying control theory to step-size controllers. A relatively recent paper in this area is by Soderlind [9]. In it, he analyses and develops an array of different controllers, building on previous work. These include controllers that provide smooth step-size sequences, have a high order of adaptivity (desired for smooth problems) and suppress high order frequencies (for nonsmooth and stochastic problems). Note that Soderlind also develops and analyses controllers that suppress high order frequencies in the error sequence but we do not test these in our work.

1.3 Report Outline

In Chapter 2, the code-coupling problem is defined in mathematical terms and explained, with the help of an example. In particular, the LD and the LDIC methods and constant and linear extrapolation for measuring deviation are all defined mathematically and explained using a running example. Also, several related terms are defined. Finally, the drawbacks of the LD and LDIC methods are discussed.

In Chapter 3, first, the elementary step-size controller is introduced. Then, the "general third-order controller" formula, which encompasses a wide array of controllers, is described. Next, step-size limiters are explained. Finally, the step-size controllers and limiters are adapted for the LD and LDIC methods, thereby creating *window-size* controllers and limiters.

In Chapter 4, we first describe the canned data testing method, the test system, and the test cases. Then we explain the plots used to visualize the results and the parameter values used for the tests. After that, we discuss the computer programs developed and used for testing and plotting. Finally, the test results are presented and analysed. The analysis includes, but is not limited to, a comparison of the performance of the window-size controllers against each other, noting the differences in the results for LD versus LDIC and CE versus LE and explaining the undesired behaviour of some controllers for some of the test cases.

In Chapter 5, implementation options are discussed and compared. By "implementation option", we mean the following: a way of implementing the LD/LDIC methods for the code-coupling problem given a choice of CE or LE and a choice of window-size controller. In particular, we focus on the degree of centralization versus distribution of the required tasks between the codes and the central window-size controller program, which is explained in Chapter 2.

In Chapter 6, we draw some conclusions from this paper and discuss possible future research directions.

Finally, in the appendices, we include some technical material that was not included in the main body of the paper. We end with a brief bibliography.

Chapter 2

Limiting Deviation Method

Before reading this chapter, recall the definitions for two terms from the Introduction: *multi-component system* and *code*. A multi-component system is a system that consists of distinct components (e.g., multi-physics systems). In a simulation of a multi-component system using operator splitting (OS), the separate, possibly concurrently running, computer programs (or program segments), each modelling a different component of the multi-component system, are referred to as codes.

Each code of a multi-component system has one or more *state variables*. In a physical system, possible state variables are temperature, pressure and volume. A set of values for the state variables of a code at a particular time constitutes the state of the subsystem corresponding to that simulation at that time. The state at a future time is determined by the present state, the governing equations of the subsystem's model and the present state of one or more of the other subsystems. State variables are simply referred to as variables here onwards.

Some code variables are used internally only, whereas others are shared between codes. Variables in the second category are referred to as *coupled variables* in this document. Specifically, the coupled variables of a code c differ from the other variables of c in that their values are provided, at each data exchange time-point, to those other codes that require them. These other codes require the values of one or more of c's coupled variables (and possibly the values of the coupled variables of other codes as well) to compute the future state of their respective subsystems. We refer to the time interval between the exchanging of the values of coupled variables as a *window*.

Concepts in the document are explained through a running example presented below. The example uses the following system: there are two codes, Code A and Code B; two of Code A's variables, x and y, are coupled with Code B and one of Code B's variables, z, is coupled with Code A. This is represented pictorially in Figure 2.1 below.

Let $[T_0, T_1]$, $[T_1, T_2]$, $\cdots$, $[T_{M-1}, T_M]$ be M windows over the interval S_{start} to S_{end} (with $T_0 = S_{start}$ and $T_M = S_{end}$), where S_{start} and S_{end} are the simulation start and end times respectively, $T_0 < T_1 < \cdots < T_M$ and T_0, T_1, $\cdots$, T_M are time-points that mark boundaries of windows. Then, at the end of each window $[T_{m-1}, T_m]$, i.e., at time T_m, Code A transfers to Code B the values of $x(T_m)$ and $y(T_m)$ and likewise, Code B transfers to Code A the value of $z(T_m)$.

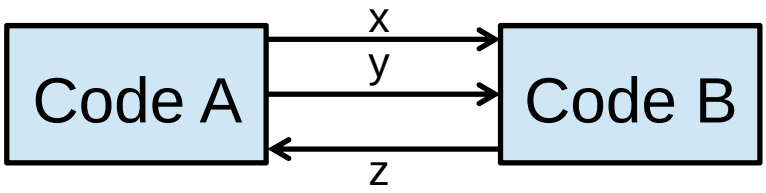

Figure 2.1: Example system consisting of two codes with three coupled variables in total.

All codes are discrete solvers, i.e., they compute the values of their variables for only a finite number of time-points in each window. These time-points include the window endpoint but not the window starting point (the values at the window starting point don't need to be computed as they are provided as the initial condition for the window). All time-points excluding the window endpoint are referred to as *internal time-points* because they lie strictly inside the window. They are adaptively determined in each code to meet code-specific accuracy constraints and, so, their spacing is not necessarily even. Also, each code may have a different set of internal time-points for the same window. To be more precise, for the n^{th} window, $[T_{m-1}, T_m]$ (n may be greater than m as some windows may be "rejected", which is explained later), the internal time-points for code c are represented by t_1^c, t_2^c, $\cdots$, $t_{l_n^c-1}^c$, where $T_{m-1} < t_1^c < t_2^c < \cdots < t_{l_n^c-1}^c < T_m$ and $l_n^c - 1$ is the number of internal time-points in the n^{th} window, $[T_{m-1}, T_m]$, for code c. Finally, the phrase "time-points of the window" refers to all l_n^c of code c's time-points in the n^{th} window, $[T_{m-1}, T_m]$ (i.e., the internal time-points and the window endpoint, $t_1^c, t_2^c, \cdots t_{l_n^c-1}^c, t_{l_n^c}^c = T_m$). We sometimes also use $t_0^c = T_{m-1}$, but, according to our definition above, t_0^c is not counted as a time-point of window $[T_{m-1}, T_m]$.

In addition to the codes described above, there is a *controller program*, whose job it is to accept or reject windows as they are completed and to determine the next window's size after the completion of each window. Note that the controller program is separate from the codes, but, of course, can communicate with them. To be more specific, each window $[T_m, T_{m+1}]$ may be accepted or rejected by the controller program. If rejected, all codes return to time-point T_m and execute a smaller window $[T_m, T'_{m+1}]$. If accepted, all codes proceed to execute the next window $[T_{m+1}, T_{m+2}]$. We use n as the index for the windows attempted (rejected and accepted) and m as the index for the accepted windows' endpoints. Regardless of whether the last window was accepted or rejected, the size of the next window is determined by the controller program in an adaptive fashion, as explained in Chapter 3.

In the simplest version of coupling, which we call *constant extrapolation*, a code uses the window-endpoint value of a coupled variable from the previous window as constant throughout the next window (we assume that the previous window was accepted by the controller program). So, in the running example, the approximate values of x and y in Code B would be $\tilde{x}(t) = x(T_{m+1})$ and $\tilde{y}(t) = y(T_{m+1})$, respectively, for all $t \in [T_{m+1}, T_{m+2}]$. Likewise, the approximate value of z in Code A would be $\tilde{z}(t) = z(T_{m+1})$ for all $t \in [T_{m+1}, T_{m+2}]$. This is represented pictorially in Figure 2.2 below, using just one coupled variable, namely Code A's x.

A more sophisticated version of coupling is based on *linear extrapolation*. Suppose that the controller program has accepted the window $[T_m, T_{m+1}]$ and that the codes are about to execute window $[T_{m+1}, T_{m+2}]$. Each code does linear extrapolation to approximate the values of the coupled variables of other codes that it needs at its internal time-points for the window $[T_{m+1}, T_{m+2}]$. It does so using the values of these coupled variables at T_{m+1} that it received after the execution of the previous window, i.e., $[T_m, T_{m+1}]$, the values at T_m that it received two windows back, i.e., after the execution of $[T_{m-1}, T_m]$,

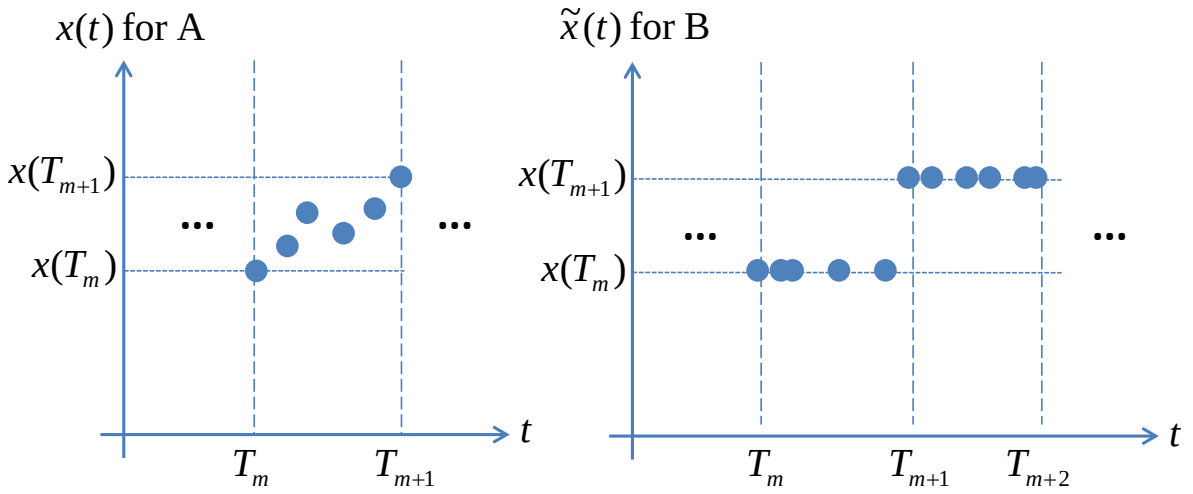

Figure 2.2: In the simplest version of coupling, the window endpoint value of a coupled variable that is transferred is used as a constant over the next window by the receiving codes. The above figure pictorially represents this for the running example using just coupled variable x: $\tilde{x}(t) = x(T_{m+1})$ is used by Code B to approximate $x(t)$ at its internal time-points in the window $[T_{m+1}, T_{m+2}]$, where $x(T_{m+1})$ is the value of Code A's x at the end of the window $[T_m, T_{m+1}]$. Similarly, $\tilde{x}(t) = x(T_m)$ is used by Code B to approximate x(t) at its internal time-points in the window $[T_m, T_{m+1}]$. Note: it is assumed that windows $[T_{m-1}, T_m]$ and $[T_m, T_{m+1}]$ were accepted by the controller program.

and finally, $T_{m+1} - T_m$. Put in terms of the running example and using only the coupled variable x, this version of coupling works as follows: after the execution of $[T_m, T_{m+1}]$ (assume it was accepted), Code A transfers to Code B $x(T_{m+1})$; Code B then computes the slope

$$s = \frac{x(T_{m+1}) - x(T_m)}{T_{m+1} - T_m} \tag{2.1}$$

where $x(T_m)$ is the value of x received by Code B after the execution of $[T_{m-1}, T_m]$ (assume it was accepted); Code B then uses the linear function

$$\tilde{x}(t) = x(T_{m+1}) + s(t - T_{m+1}) \tag{2.2}$$

to approximate $x(t)$ at all its internal time-points in the window $[T_{m+1}, T_{m+2}]$. This is depicted pictorially in Figure 2.3 below. The same would be done by Code B for y and by Code A for z.

Note that linear extrapolation can be used for all the windows of the simulation, except for the first window where the value $x(T_m)$, the window endpoint value of the window two windows back, is not available (the "window endpoint value" of the "previous window" is available in the form of the initial value). If the problem's nature changes radically after window $[T_m, T_{m+1}]$, it may be better to switch to constant extrapolation at that point, since $x(T_m)$ and $x(T_{m+1})$ may not be representative of the solution in the window $[T_{m+1}, T_{m+2}]$. However, this point is not explored in this work.

The procedure described above is based on linear extrapolation. Higher-order polynomial extrapolation could also be used, although we do not pursue this option in this work.

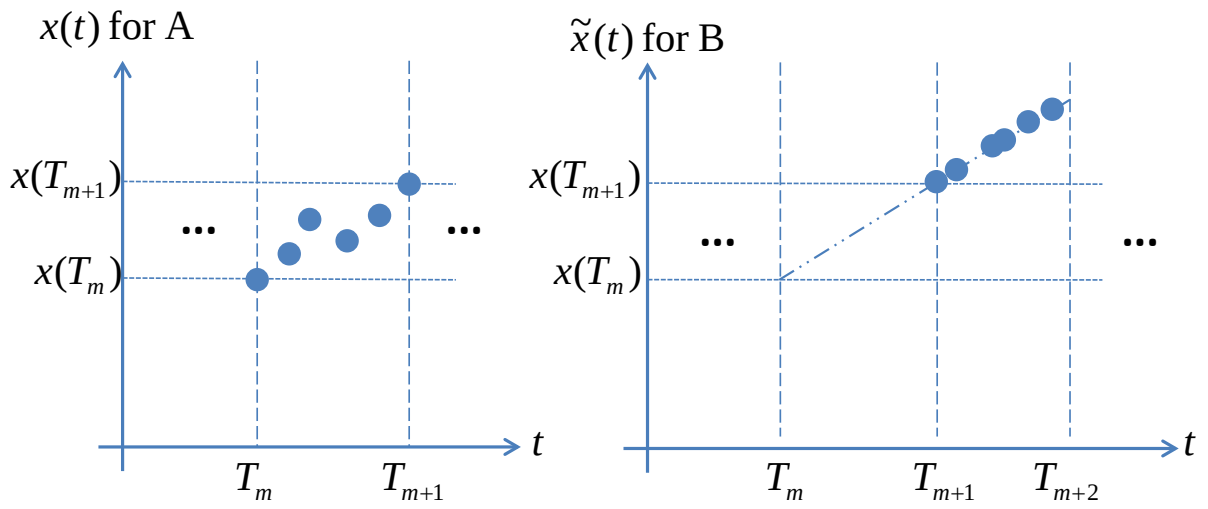

Figure 2.3: A more sophisticated version of coupling, linear extrapolation, is used by Code B to approximate x at its internal time-points in the window $[T_{m+1}, T_{m+2}]$ using $x(T_{m+1})$, $x(T_m)$ and $T_{m+1} - T_m$. $\tilde{x}(t) = x(T_{m+1}) + s(t - T_{m+1})$ is used by Code B to approximate $x(t)$ at all its internal time-points in the window $[T_{m+1}, T_{m+2}]$, where $s = \frac{x(T_{m+1}) - x(T_m)}{T_{m+1} - T_m}$ and $x(T_m)$ and $x(T_{m+1})$ are the values of Code A's x at the end of the windows $[T_{m-1}, T_m]$ and $[T_m, T_{m+1}]$, respectively. Note: it is assumed that windows $[T_{m-1}, T_m]$ and $[T_m, T_{m+1}]$ were accepted by the controller program.

Since the coupled variables are updated only at window start/end points rather than at every internal time-step, error is introduced into the computation due to using inaccurate values for the coupled variables. This error is referred to as *windowing error* in this document. The purpose of the controller program is twofold. First, it rejects/accepts windows as the simulation progresses to try to ensure that the global result of the simulation is satisfactorily accurate. Second, it adaptively adjusts the window-sizes so as to improve efficiency (if a larger window meets the accuracy requirements, it is preferred over a smaller window).

One of the strategies the controller program may use for controlling the windowing error is the Limiting-Deviation (LD) method (name given by us). Given certain assumptions, the LD method can indirectly control the windowing error. The basic idea of the method is to limit the deviation of every coupled variable in the system over each window w so that the approximate values that were used by other codes for w aren't too different from the actual values. It rejects a window if any one of the coupled variables' deviation is larger than what is deemed to be tolerable for that variable. There are two possible variations of the LD method for the operator-splitting code-coupling problem. The first limits the deviations of the coupled variables at the endpoint of each window and the second limits the deviations throughout each window, i.e., at each of the internal time-points of the window as well as the window endpoint. We call the first variation the LD method and the second, the LD with interior check (or LDIC) method.

The deviation can be measured in terms of an absolute difference, a relative difference or some combination of relative and absolute difference. In the examples below, the LD and LDIC methods use absolute difference (as is explained below).

We illustrate the LD and LDIC methods in terms of the running example using only Code A's cou-

pled variable x and constant extrapolation. In the LD method, code A checks[1] if $|x(T_{m+1}) - \tilde{x}(T_{m+1})| = |x(T_{m+1}) - x(T_m)| \leq tol_x$ (where $|\cdot|$ is the absolute value) for every window $[T_m, T_{m+1}]$. Remember that $\tilde{x}(t) = x(T_m)$ is the constant value used by Code B to approximate $x(t)$ over the window $[T_m, T_{m+1}]$. This is pictured on the left side of Figure 2.4 below. In the LDIC method, Code A checks, for every window $[T_m, T_{m+1}]$, if $|x(t_j) - \tilde{x}(t_j)| = |x(t_j) - x(T_m)| \leq tol_x$ for all j, where $t_j \in [T_m, T_{m+1}]$ are the internal time-points for which Code A generated values for x; it also checks if $|x(T_{m+1}) - \tilde{x}(T_{m+1})| = |x(T_{m+1}) - x(T_m)| \leq tol_x$. This is pictured on the right side of Figure 2.4 below.

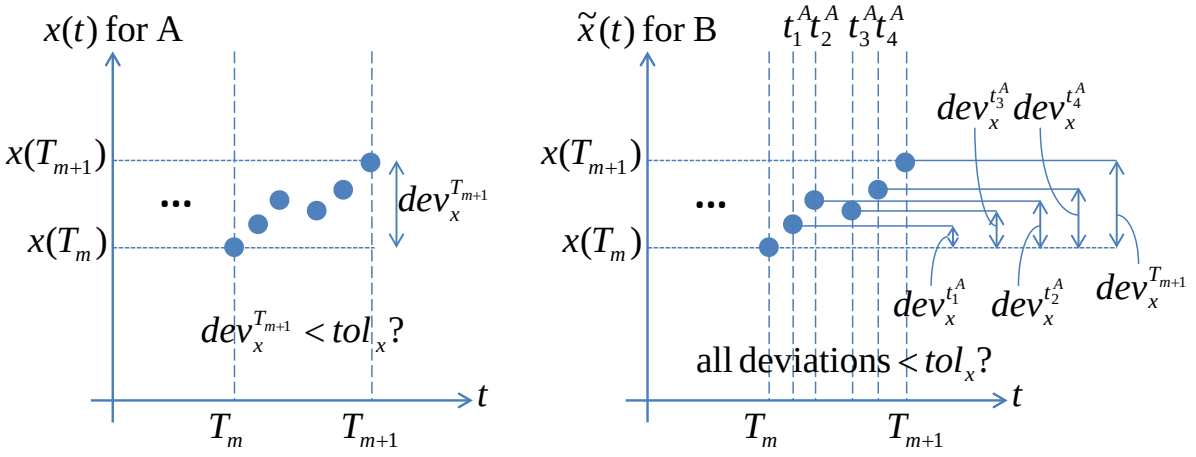

Figure 2.4: The plot on the left illustrates the LD method with constant extrapolation applied to Code A's coupled variable x in the window $[T_m, T_{m+1}]$. The plot on the right is the same except with the LDIC method instead of LD. The LD method with constant extrapolation effectively requires that the deviation across the window endpoints of every coupled variable be less than that variable's tolerance. On the other hand, the LDIC method requires that the deviations with respect to the window starting point of every coupled variable at all the internal time-points and the window endpoint be less than that variable's tolerance.

If all the deviations checked are less than the corresponding tolerances, then the window is accepted and the simulation continues from T_{m+1}. Otherwise, the window is rejected and a smaller new window is attempted starting from T_m. In either case, the next window's size is determined as described in Chapter 3.

The benefit of doing internal checks in the LDIC method is that there may be a significant change in value within the window for one or more coupled variables. For example, a window may contain a spike.

The LD and LDIC methods with linear extrapolation are similar to the LD and LDIC methods, respectively, with constant extrapolation, which was described above. For the LD method, code A checks that $|x(T_{m+2}) - \tilde{x}(T_{m+2})| = |x(T_{m+2}) - (x(T_{m+1}) + s(T_{m+2} - T_{m+1}))| \leq tol_x$, where $s = \frac{x(T_{m+1}) - x(T_m)}{T_{m+1} - T_m}$. For the LDIC method, code A checks that $|x(t_j) - \tilde{x}(t_j)| = |x(t_j) - (x(T_{m+1}) + s(t_j - T_{m+1}))| \leq tol_x$ for all the internal time-points t_j in the window $[T_{m+1}, T_{m+2}]$ as well as the endpoint T_{m+2}, where again $s = \frac{x(T_{m+1}) - x(T_m)}{T_{m+1} - T_m}$.

[1] In this section, we assume the check is performed by code A, but the check could be performed in the controller program, as discussed in Chapter 5.

It has been assumed so far that all the coupled variables are one-dimensional quantities, but, of course, they may be multi-dimensional. If a system has one or more multi-dimensional coupled variables, simply replace $|\cdot|$ with $\|\cdot\|$ when computing deviations for these coupled variables, where $\|\cdot\|$ is an appropriate norm. Similarly, instead of bounding coupled variables, such as code A's x and y, separately, we could concatenate them into one large vector (e.g., $v = (x, y)$) and bound some norm of the deviation associated with that vector.

The LD and LDIC methods have several disadvantages. First, deviations don't necessarily imply local windowing error. That is, supposing LD is used, it is possible for the endpoint deviation of a window to be greater than the prescribed deviation tolerance while the local error of the window endpoint is less than the local error tolerance that would achieve the desired accuracy. For LDIC, the same is true and it applies to internal time-points too. As a result, the sizes of some windows may be smaller than they need to be. This results in a greater number of windows, which increases the amount of data transferred (the amount of data transferred increases linearly with the number of windows) and the amount of computation (while this is true, this increase is small compared to the increase brought on by extra rejected windows (provided a particular assumption holds, as explained in §4.6.3); however, with enough windows, this increase also becomes significant).

Second, the tolerances for the deviations that limit the local errors to the desired values must be determined by running multiple simulations and knowing a sufficiently accurate approximation to the true solution. Third, the deviation tolerances required to limit the local errors to the same desired values in a different simulation may be different! So, one may have to redo the test simulations to determine the correct deviation tolerances for the new simulation.

Despite these disadvantages, we evaluate the LD and LDIC methods in this paper, because they are easily implemented strategies that the scientists at AECL believe will be effective for their application.

Chapter 3

Adaptive Window-Size Selection

We develop an adaptive window-size selection algorithm in this chapter. We begin by reviewing adaptive step-size selection algorithms for initial value problems (IVPs) for ordinary differential equations (ODEs), since these two problems are similar and step-size selection algorithms for IVPs of ODEs have been studied intensively.

3.1 Step-Size Controllers for ODE Solvers

IVPs for ODEs can be solved *numerically* using one or more computers. A numerical solution of a system of ODEs is *discrete*, i.e., the solution is approximated at a finite number of points in the domain. Supposing the domain of the problem is time, the solution is obtained by taking steps in time over the domain starting from the domain starting point and progressing to the domain ending point. These steps are called *time-steps*. The size of each time-step is adaptively determined so as to take as large a step as possible while still meeting the local accuracy constraints. Solution values are computed for the endpoint of each time-step by a *numerical method*. These solution values along with the corresponding time-points make up the discrete approximation to the solution for the system of ODEs. Refer to [4], for example, for the basics of numerically solving IVPs for ODEs.

The following is a description of the context in which step-size controllers are used in numerically solving IVPs for ODEs. To simplify the discussion below, suppose that instead of a system of ODEs, we have just a single ODE, i.e., the solution has just a single component. Then, for each time-step, after the solution value at the end of the time-step is computed, an estimate of the error in the solution value due to that step is compared to the local error tolerance. If the error estimate computed is less than or equal to the tolerance, that step is accepted. On the other hand, if the error estimate computed is greater than the tolerance, that step is rejected. If the step is rejected, the next step starts from where the old step started. Regardless of whether the step is accepted or rejected, the size of the next time-step is calculated using a *step-size controller*.

The simplest of the step-size controllers is the so-called *elementary controller*, which we outline in §3.1.1 below. Following that, the *general third-order controller* is presented in §3.1.2.

Note that the controller formulae presented below apply only when the ODE problem has one solution component, since our window-size controllers are described for single-component coupled variables. However, the techniques described below for the step-size controller can be generalized to multi-component variables (i.e., vectors) by replacing the absolute value of the error by a suitable norm.

3.1.1 Elementary Controller

The elementary controller is given by the formula

$$h_{n+1} = \left(\frac{\gamma \cdot tol}{err_n}\right)^{\frac{1}{k}} h_n \tag{3.1}$$

where

h_{n+1} is the new step-size

h_n is the previous step-size

err_n is the absolute value of the error estimate for the previous step

tol is the error tolerance (note $tol > 0$)

γ is the safety factor, typically greater than or equal to 0.9 and always less than 1

k controls the degree of adaptive change (note $k > 0$).

Formula (3.1) works as follows. Let $\alpha = \left(\frac{tol}{err_n}\right)^{\frac{1}{k}}$ and assume for now that $\gamma = 1$. The new step-size, h_{n+1}, will be larger than the previous step-size, h_n, if err_n, the error estimate for the previous step, is less than the tolerance, since α in this case is greater than 1. Moreover, for a fixed tolerance, α is larger for a smaller error estimate. Thus, the degree to which the new step-size becomes larger than the previous one depends on how much smaller the error estimate is than the tolerance. Likewise, the new step-size, h_{n+1}, will be smaller than the previous step-size, h_n, if the error estimate is greater than the tolerance, since α in this case is less than 1. Moreover, for a fixed tolerance, α is smaller for a larger error estimate. Thus, the degree to which the new step-size becomes smaller than the previous one depends on how much larger the error estimate is than the tolerance. The safety factor γ, which is always less than 1 and usually greater than or equal to 0.9, makes the new step-size, h_{n+1}, conservative by making the multiplicative factor $\left(\frac{\gamma \cdot tol}{err_n}\right)^{\frac{1}{k}}$ in (3.1) smaller than α.

Please refer to Appendix A.1 for the derivation of (3.1), the elementary step-size controller formula.

In the context of ODEs, it is usual to take $k = p$ or $k = p+1$, where p is the order of the underlying ODE method. Whether $k = p$ or $k = p+1$ depends on whether the controller is attempting to satisfy an *error per unit step* or *error per step* criterion, respectively. In the case of ODEs, the underlying method could be a Runge-Kutta, Taylor Series or linear multistep method, to name just a few possible methods.

3.1.2 General Third-Order Controller

The elementary step-size controller (3.1) uses only the previous step's information to compute the next step's size. However, one may also use information from the steps before the previous step. Indeed, there are many well-known controllers, such as the PI and PID controllers, that use more information from the past than just the previous step's. Formula (3.2) below is the general form of a wide class of controllers that use information from *three* prior steps.

$$h_{n+1} = \left(\frac{\gamma \cdot tol}{err_n}\right)^{\beta_1} \left(\frac{tol}{err_{n-1}}\right)^{\beta_2} \left(\frac{tol}{err_{n-2}}\right)^{\beta_3} \left(\frac{h_n}{h_{n-1}}\right)^{-\alpha_2} \left(\frac{h_{n-1}}{h_{n-2}}\right)^{-\alpha_3} h_n \tag{3.2}$$

where β_1, β_2, β_3, α_2 and α_3 are parameters, the values of which must be chosen, and

tol is the error tolerance (note $tol > 0$)

γ is the safety factor, typically greater than or equal to 0.9 and always less than 1

h_n is the size of the n^{th} step

err_n is the absolute value of the error estimate for the n^{th} step.

Formula (3.2) encompasses controllers that use fewer than three prior steps, e.g., the elementary controller and the well-known PI and PID controllers. That is, it can be reduced to the formulae for these controllers by choosing appropriate α and β parameter values (e.g., choosing $\beta_2 = \beta_3 = \alpha_2 = \alpha_3 = 0$ and $\beta_1 = 1/k$ reduces (3.2) to the elementary controller, (3.1)).

Formula (3.2) is said to have *third-order dynamics*, hence the name ascribed to it in this document: the *general third-order controller*. The meaning of *order of dynamics* is rooted in control theory. A discussion of the control theoretic differences between the controllers is omitted in this document. Rather, just the behavioural differences between the various controllers is discussed. One may refer to [2] for an introduction to control theory and [8] for an introduction to the control theoretic analysis of step-size controllers. As well, one may refer to [9] for a complete control theoretic analysis of the various controllers presented and tested here.

The order of dynamics of a controller, denoted p_D, is equal to the number of past steps used by the controller. So, $p_D = 1$ for (3.1). If $\beta_3 = \alpha_3 = 0$ in (3.2) and all other α and β parameters are non-zero, except possibly for *one* of β_2 and α_2, then $p_D = 2$. One of β_2 and α_2 can be zero because information from the step immediately before the previous step is still used. Likewise, for a $p_D = 3$ controller, $\beta_1 \neq 0$, both β_2 and α_2 cannot be 0 and both β_3 and α_3 cannot be 0.

All controllers are characterized by three values: p_D, p_A and p_F. The value p_D has already been explained. The value p_A is the *order of adaptivity*. It is a measure of the rate at which the local error estimate is adapted to the tolerance. The value p_F is the *filter order*. It is a measure of the *regularization* effect that the controller has on the step-size sequence. A sequence A is more regular than a sequence B if A is smoother than B. A more complete discussion of the order of adaptivity and the filter order is given in [9].

Note that the following must always hold (see [9] for an explanation based on control theory):

$$p_A + p_F \leq p_D \tag{3.3}$$

The controllers are divided into two categories: *deadbeat controllers* and *non-deadbeat controllers*. Deadbeat controllers, to quote Söderlind [9], are "suitable only for very smooth problems, and also put stringent demands on how supporting algorithms, such as equation solvers, are implemented." The supporting algorithms are numerical methods such as linear or non-linear equation solvers, used within the ODE solving method. Non-deadbeat controllers, on the other hand, can better handle problems that aren't smooth and are less demanding of the supporting algorithms. Following the labelling system used in [9], controllers are denoted by $Hp_Dp_Ap_F$ for non-deadbeat controllers (e.g., $H110$) and $H_0p_Dp_Ap_F$ for deadbeat controllers (e.g., H_0110). See [8, 9] for a more complete discussion of deadbeat and non-deadbeat controllers (refer to [2] for an introduction to control theory if needed).

For each of the various combinations of p_D, p_A and p_F (with $p_D \leq 3$) that satisfy (3.3), the values of the α and β parameters that make the controller a deadbeat controller are provided in Table 3.1 below (values obtained from [9]). Note that, instead of providing values for β_1, β_2 and β_3, Söderlind [9] provides values for $k\beta_1$, $k\beta_2$ and $k\beta_3$, where $k = p$ or $k = p + 1$ and p is the order of the underlying numerical ODE method. Also note that, in Table 3.1, a "-" is used for a parameter that is necessarily zero. For example, for all second-order controllers, β_3 and α_3 are necessarily zero. (The reason for not simply using 0 is to distinguish parameters that are necessarily zero from those that are zero by choice. For deadbeat controllers, there is no choice in the parameter values, so the convention is not useful here. However, for non-deadbeat controllers, presented later, there is and, so, this convention is useful there. The convention is used here also for consistency with the corresponding non-deadbeat controllers listed in Table 3.2.)

Table 3.1: Parameters of $p_D \leq 3$ Deadbeat Controllers

$k\beta_1$	$k\beta_2$	$k\beta_3$	α_2	α_3	p_D	p_A	p_F	Name
1	-	-	-	-	1	1	-	H_0110
2	-1	-	-1	-	2	2	0	H_0220
1/2	1/2	-	1/2	-	2	1	1	H_0211
3	-3	1	-2	1	3	3	0	H_0330
5/4	1/2	-3/4	-1/4	-3/4	3	2	1	H_0321
1/4	1/2	1/4	3/4	1/4	3	1	2	H_0312

For the non-deadbeat controllers, Söderlind [9] suggests values for the α and β parameters that are desirable. Again, he provides values for $k\beta_1$, $k\beta_2$ and $k\beta_3$, instead of β_1, β_2 and β_3. The parameter values for the various non-deadbeat controllers are provided in Table 3.2 below. Also, in Table 3.2, a "-" is used for a parameter that is necessarily zero and 0 is used for a parameter that is made zero by choice. For example, for $H211PI$, β_3 and α_3 are necessarily zero since it is a second-order controller and β_2 is 0 by choice in order to make the controller PI.

Table 3.2: Parameters of $p_D \leq 3$ Non-Deadbeat Controllers

$k\beta_1$	$k\beta_2$	$k\beta_3$	α_2	α_3	Name
1/b	1/b	-	1/b	-	H211b
1/6	1/6	-	0	-	H211 PI
1/b	2/b	1/b	3/b	1/b	H312b
1/18	1/9	1/18	0	0	H312 PID
1/3	1/18	-5/18	-5/6	-1/6	H321
3/10	1/20	-1/4	-1	0	H321 Predictive PID

Note that, for two of the controllers in Table 3.2, a single variable b determines the parameter values instead of them all being fixed. For these, Söderlind [9] suggests using values from a certain range and explains the benefits/drawbacks of choosing a particular value in that range. For $H211b$, $b \in [2, 8]$ is suggested for practical use, with the larger values in the range providing increased smoothness in the step-size and error sequences, but, also, larger low-frequency errors. The particular value of $b = 4$ is recommended based on numerical experiments. For $H312b$, the situation is similar, except that $b \in [4, 16]$ is suggested and $b = 8$ is recommended.

Also note that, for three of the controllers in Table 3.2, the controller name has an extra word or words appended. The reason is as follows. If, in (3.2), $\alpha_2 = \alpha_3 = 0$ and $\beta_i \neq 0$ for $i = 1, 2, 3$, the resulting controller is a member of the special, well-known class of *PID controllers*. If, $\beta_3 = \alpha_2 = \alpha_3 = 0$ and β_1 and β_2 are non-zero, the resulting controller is a member of the special, well-known class of *PI controllers*. Finally, a PI controller with $\alpha_2 = -1$ is referred to as a predictive PI controller and a PID controller with $\alpha_2 = -1$ is referred to as a predictive PID controller. Predictive PI and PID controllers achieve $p_A = 2$, whereas the other PI and PID controllers do not.

3.1.3 Step-Size Limiters

It is common in numerical methods for ODEs to place limits on the increase and decrease of the step-size over a single step [8]. As well, there may be both *relative* and *absolute* limits. The relative limits are based on the previous step's size. The relative limits are imposed first and then the absolute limits. The formula for imposing the relative limits is

$$h'_{n+1} = \min\{\max\{h_{n+1}, 0.5h_n\}, 2h_n\} \tag{3.4}$$

where

h_n is the previous step's size

h_{n+1} is the step-size computed by the step-size controller

0.5 and 2 are the lower and upper relative limit factors respectively - they can be changed to any other appropriate values

h'_{n+1} is the step-size after the relative step-size limits are imposed

The formula for imposing the absolute limits is

$$h''_{n+1} = \min\{\max(h'_{n+1}, h_{min}), h_{max}\} \tag{3.5}$$

where

h'_{n+1}	is the step-size after the relative step-size limits are imposed
h_{min} and h_{max}	are the absolute lower and upper bounds respectively on the step-size that are reasonable to use for the problem on the particular computing system
h''_{n+1}	is the step-size after the absolute step-size limits are imposed

3.2 Modifications to the Standard Use of Step-Size Controllers

3.2.1 Window-Size Selection after Rejected Windows

For certain values of the α and β parameters, the general third-order controller (3.2) has the following undesirable property: the next step's size may be larger than the previous step's size even if the previous step was rejected. However, the size of the next step should always be reduced if the previous step was rejected (assuming that the same underlying ODE method is used on both steps). So, in our use of the general controller, (3.2), i.e., for the coupling-codes problem, after the first rejected window, H_n, following one or more accepted windows, we determine the size of the next window, H_{n+1}, using

$$H_{n+1} = \min\{\ \text{output of } X,\ \ \text{output of } H_0110,\ \ 0.9H_n\ \} \tag{3.6}$$

where X is the controller being used.

With (3.6), the next window's size is guaranteed to be reduced (by a factor of at least 0.9) even if the output of controller X is greater than the size of the previous window, H_n. Note that we could have chosen another way to compute H_{n+1} that guarantees that $H_{n+1} < H_n$ after a rejected window. However, we chose (3.6) for the following reasons. Controller X may provide a good new window-size. If there had been multiple successive rejections prior to this rejection, one should abandon the controller, since the asymptotic error model may not be valid in this region. However, there has been just one so far. Hence, we do not abandon the controller yet. The term $0.9H_n$ is included in (3.6) to guarantee a reduction in the window-size by at least a factor of 0.9. Finally, we include the elementary controller as an added measure of conservativeness: if it's prediction is smaller than that of controller X (and $0.9H_n$), it is used, even though controller X's prediction may be better.

If H_{n+1} is also rejected, we use $\frac{H_{n+1}}{2}$ for the size of the next attempted window. The reason we abandon controller X and H_0110 is that the asymptotic error model may not be valid in this region. The sizes of all the windows that follow subsequent rejections will also be obtained by halving the previous window's size. That is, we use the following:

$$H_i = \frac{H_{i-1}}{2} \text{ for } i \geq n+2 \tag{3.7}$$

Note that we could have chosen another way to compute H_i for $i \geq n+2$. We chose (3.7) because controller X is clearly not working well at this point and (3.7) is a simple way to reduce the window-size fairly quickly.

3.2.2 Window-Size Selection after the First Accepted Window following One or More Rejections

Another modification to the standard use of step-size controllers for windowing systems is we do not use the sizes of previous rejected windows to compute the next window's size, after an accepted window. The reason is that the rejection(s) may be due to a rapid change in one or more of the variables of the system, and as such, information from the previous rejected windows will not be useful in making a good prediction for the size of the next window. So, after the first accepted window after one or more consecutive rejected windows, the following should be done:

- if $p_D = 1$, nothing special is required: just continue to use the chosen $p_D = 1$ controller;
- if $p_D = 2$, use a $p_D = 1$ controller to compute the next window's size. If that window is accepted, continue with the $p_D = 2$ controller that was being used before;
- if $p_D = 3$, use a $p_D = 1$ controller to compute the next window's size. If that window is accepted, use either a $p_D = 1$ or a $p_D = 2$ controller to compute the next window's size (a $p_D = 2$ controller is preferred as it will likely provide a better prediction than a $p_D = 1$ controller). If that is accepted as well, continue with the $p_D = 3$ controller that was being used before.

For all three cases, if there is a rejection, apply the computing-window-sizes-after-rejections policy described in §3.2.1 until a window is accepted. Then, repeat the instructions for that case.

3.3 Adaptive Window-Size Selection for the LD Method

We now extend the adaptive step-size selection algorithms described in §3.1 to adaptive window-size selection algorithms. We begin by considering the LD method, which is based on the formula

$$H'_{n+1} = \min \left\{ \left(\frac{\gamma \cdot tol^{i,c}}{dev_n^{i,l_n^c,c}} \right)^{\beta_1} \left(\frac{tol^{i,c}}{dev_{n-1}^{i,l_{n-1}^c,c}} \right)^{\beta_2} \left(\frac{tol^{i,c}}{dev_{n-2}^{i,l_{n-2}^c,c}} \right)^{\beta_3} \left(\frac{H_n}{H_{n-1}} \right)^{-\alpha_2} \left(\frac{H_{n-1}}{H_{n-2}} \right)^{-\alpha_3} H_n \right. \tag{3.8}$$
$$\left. : \ i = 1, ..., r^c \text{ and } c = 1, ..., C \right\}$$

$$H''_{n+1} = \min \left\{ \max \left\{ H'_{n+1}, 0.5H_n \right\}, 2H_n \right\} \tag{3.9}$$

$$H_{n+1} = \min\left(\max\left(H''_{n+1}, H_{min}\right), H_{max}\right) \tag{3.10}$$

where

H_n	is the previous window-size
r^c	is the number of coupled variables computed by code c
i	is the index for the i^{th} coupled variable computed by code c
C	is the number of codes in the system
$dev_n^{i,l_n^c,c}$	is the deviation of the n^{th} window's $l_n^{c\,\text{th}}$ time-point (i.e., the endpoint) for the i^{th} coupled variable computed by code c
$tol^{i,c}$	is the deviation tolerance for the i^{th} coupled variable computed by code c
γ	is the safety factor, typically greater than or equal to 0.9 and always less than 1
H'_{n+1}	is the minimum among the window-sizes adaptively determined for all the coupled variables
H''_{n+1}	is the result of applying the relative limits on H'_{n+1}. That is, H''_{n+1} is H'_{n+1} unless $H'_{n+1} < 0.5H_n$ or $H'_{n+1} > 2H_n$. In the former case H''_{n+1} is set to $0.5H_n$ and in the latter case H''_{n+1} is set to $2H_n$
H_{min}	is the window-size absolute minimum limit
H_{max}	is the window-size absolute maximum limit
H_{n+1}	is the final value for the new window-size, obtained by imposing a second set of limits, which are absolute

The curly braces in (3.8) contain the results of applying the step-size controller formula (3.2) to all the coupled variables in the system for the endpoint of the window under consideration. The right side of (3.8) then evaluates to the minimum among these values. The reason that the new window-size is determined in this way is as follows. For each coupled variable, the controller formula *predicts*, using the window(s) just completed, the size for the next window that is as large as possible while still satisfying the deviation constraint. So, assuming that the controller's predictions are correct, choosing the minimum among these values implies that *all* the coupled variables will satisfy their deviation constraints in the next window and consequently, the next window will be accepted. (Remember however that the step-size controller formula produces predictions, not accurate forecasts. So, it is possible that the next window will be rejected even if we choose the minimum value for the next window-size. Likewise, it is possible that the next window will be accepted even if we choose a window-size larger than the minimum.)

Formula (3.9) places relative limits on the next window's size based on the previous window's size H_n. The factors 0.5 and 2 can be changed to any other appropriate values. Formula (3.10) places absolute limits on the next window-size. The relative limits based on H_n are useful since they prevent a drastic change in the window-size from one window to the next, which would prevent certain possible misses of deviance beyond tolerance in the values of the coupled variables. Not using absolute limits, i.e., just using relative limits, posses the following problem: it is possible for the window-sizes to steadily increase or decrease without bound.

When using constant extrapolation, one should use $k = 1$, and when using linear extrapolation, one should use $k \in [1, 2]$, as explained in Appendix A.2 and A.3, respectively (for example, to determine the values of α and β of a particular controller given the values of $k\alpha$ and $k\beta$ for that controller in Table 3.1 or Table 3.2). We experimented with $k = 1$, $k = 2$ and some $k \in (1, 2)$ in the elementary controller formula with LE and found that the choice of k in that range does not make a significant difference.

It is important to note that the choice of k does not affect the reliability of the windowing system! No matter what k we use, if the deviation is larger than tol, the window will be rejected and we will attempt a smaller window starting again from T_i. However, the choice of k may affect the efficiency of the method, but, as noted above, we found that the efficiency is fairly insensitive to the choice of $k \in [1, 2]$.

Also note that the parameter k only affects the rate of change of the window-size selector, not whether the window-sizes will be increasing or decreasing. However, the rate of change does affect the efficiency of the windowing system.

Note that it is possible that, when attempting the next window at some point in the simulation, any window of size greater than or equal to the absolute minimum window-size, H_{min}, will be rejected. That is, it is possible that the deviations of one or more coupled variables will be larger than the corresponding tolerances for any window of size greater than or equal to H_{min}. In this case, the LD/LDIC method cannot proceed any further and the computation terminates.

Note that the LD and LDIC methods will likely reject any window that contains a discontinuity of size greater than the corresponding coupled variable's tolerance (by discontinuity, we mean an instantaneous jump/drop in value, not a steep climb/drop). There are two possibilities here, which are explained next.

In the first case, despite there being a discontinuity larger than the corresponding coupled variable's tolerance in the window, the window is accepted. This can happen with the LD method if the discontinuity lies in the interior of the window and the endpoint deviation is within the tolerance. In other words, the discontinuity is "missed" by the method (just as spikes are missed, as discussed before). For the LDIC method, this can happen if the discontinuity lies within the window, the endpoint deviation is within the tolerance *and* the interior point deviations of the top and bottom of the discontinuity are less than the tolerance.

In the second case, if there is a discontinuity larger than the corresponding coupled variable's tolerance in the window, the window will be rejected, regardless of its size. This can happen with the LD and LDIC methods if the window endpoint deviation exceeds the tolerance due to the discontinuity (e.g., a step drop in a coupled variable's value occurs within the window). After the first attempt on such a window is rejected, the window-size will of course be reduced. The second attempt may be accepted since the window could end before the discontinuity. However, all that does is move the next window's starting point closer to the discontinuity. Then, the same situation arises again. In this scenario, the simulation will eventually terminate since the window-size will be reduced repeatedly until first, the absolute lower limit, H_{min}, is reached and then, even the window of that size fails. If the window endpoint deviation does *not* exceed the tolerance (which is unlikely if there is a discontinuity within the window that is larger than the tolerance), then the LD method will accept the window (i.e., it will

"miss" the discontinuity). However, the LDIC method will still reject the window so long as the interior point deviations of the top and/or bottom of the discontinuity exceed the tolerance (in the step drop example, the deviation of the point at the bottom of the discontinuity would exceed the tolerance).

The second possibility is far more likely. That is why we say that the methods will *likely* reject any window with a discontinuity larger than the tolerance. Therefore, if discontinuities in the coupled variables are expected, a more sophisticated deviation control strategy must be used.

3.4 Adaptive Window-Size Selection for the LDIC Method

Consider the formula

$$H'_{n+1} = \min \left\{ \left(\frac{\gamma \cdot tol^{i,c}}{\max\limits_{j=1...l_n^c} \{dev_n^{i,j,c}\}} \right)^{\beta_1} \left(\frac{tol^{i,c}}{\max\limits_{j=1...l_{n-1}^c} \{dev_{n-1}^{i,j,c}\}} \right)^{\beta_2} \left(\frac{tol^{i,c}}{\max\limits_{j=1...l_{n-2}^c} \{dev_{n-2}^{i,j,c}\}} \right)^{\beta_3} \times \left(\frac{H_n}{H_{n-1}} \right)^{-\alpha_2} \left(\frac{H_{n-1}}{H_{n-2}} \right)^{-\alpha_3} H_n \ : \ i = 1, ..., r^c \text{ and } c = 1, ..., C \right\} \tag{3.11}$$

$$H''_{n+1} = min(max(H'_{n+1}, 0.5H_n), 2H_n)$$

$$H_{n+1} = \min\left(\max\left(H''_{n+1}, H_{min}\right), H_{max}\right)$$

where all terms are the same as the corresponding ones in (3.8), (3.9) and (3.10) except l_n^c, j and $dev_n^{i,j,c}$:

- l_n^c is the number of time-points in the current window for code c
- j is the index for the j^{th} time-point in the current window for code c
- $dev_n^{i,j,c}$ is the deviation of the i^{th} coupled variable computed by code c for code c's j^{th} time-point in the current window

The curly braces in (3.11) contain the results of applying the controller formula for each of the time-points of each of the coupled variables in the system for the window under consideration. The right side of (3.11) then evaluates to the minimum among these values. The two equations after (3.11) are similar to (3.9) and (3.10) in §3.3 respectively. They are restated here for convenience. The reason for their use here with LDIC is the same as that given in §3.3 for the LD method.

Chapter 4

Testing and Analysis of Results

4.1 Testing Methodology

The various step-size controllers in Tables 3.1 and 3.2 were tested against each other using a particular testing method, referred to by us as *canned data testing* (*CDT*). With CDT, each code's coupled variables' values as a function of time are predetermined, i.e., before the coupled "simulation" is run. So, the coupled variables of any one code do not affect the coupled variables of the other codes. To determine the values of its coupled variables at a given time point, each code uses linear interpolation on its table of predetermined values.

Note that instead of using CDT, one could use a multi-component system consisting of ODEs or PDEs. That is, each component of the multi-component problem would be a system of ODEs or PDEs and each code would include an ODE/PDE solver, which numerically solves the ODE/PDE subsystem associated with that code. The codes would actually exchange values of coupled variables at window endpoints and use them to compute future values for the coupled variables. This approach is preferred over CDT since it takes the coupling of components into account whereas CDT does not. However, as stated in Chapter 1, CDT is AECL's preferred approach, for this preliminary study. That is the reason for it's use in this work.

4.2 Test System

The test system employed consists of two codes: code A and code B. Code A has two coupled variables and code B has one. For this system, each controller was tested against five different test cases, once using constant extrapolation and once using linear extrapolation. The five test cases are as follows:

1. all coupled variables in the system are smoothly varying sinusoids with differing frequencies
2. coupled variable #1 of code A has a spike, while the other two coupled variables in the system are linear functions of time

3. coupled variable #1 of code A has a narrow spike, while the other two coupled variables in the system are linear functions of time

4. coupled variable #2 of code A has a steep drop, while the other two coupled variables in the system are slowly varying functions of time

5. coupled variable #1 of code A has a spike, coupled variable #2 of code A has a steep rise and coupled variable #1 of code B has a steep drop. So, this test case has a combination of features.

4.3 Plotting of Results

For each run of a test case,

- the following are plotted *together* (these plots are referred to as *transferred values of coupled variables versus time* plots in this paper)
  - the values of coupled variables #1 and #2 of code A that were "used" by code B versus time
  - the values of coupled variable #1 of code B that were "used" by code A versus time
- the window-sizes are plotted against time

The plots above are generated for both the LD and LDIC methods. Thus, there are four plots in total per test case, all of which are put in one frame (in a 2×2 grid) for convenient comparison. So, there are 10 2×2 plots for each controller (5 for constant extrapolation and 5 for linear extrapolation).

As explained in §4.6.3, the difference in "performance" (term defined in §4.6.3) between two controllers is determined primarily by the difference in their number of rejected windows, assuming that the codes do not store the values computed for the most recent window in memory (if they did, the difference in performance between two controllers is equally impacted by the difference in the number of accepted windows and the difference in the number of rejected windows). The number of accepted windows and the number of rejected windows are displayed in the window-size versus time plots described above.

The following is an explanation of the labels of the legend in the LDIC window-sizes versus time plots (the legend labels are abbreviated; so, one may need to refer to this explanation to clarify their meaning)

- "Acptd; no intr pnt/s" - the window did not contain any interior points and was accepted
- "Acptd; intr pnt/s exist" - the window did contain interior points and was accepted
- "Extr Rjctd" - the window was rejected due to the exterior point, i.e., the window endpoint, only
- "Intr Rjctd" - the window was rejected due to one or more interior time points only
- "Extr and Intr Rjctd" - the window was rejected due to both the exterior point (window endpoint) *and* one or more interior time points

4.4 Testing Parameter Values and Corner Cases

For all the test cases,

- the deviation tolerance is the same for all the coupled variables, which is displayed on the transferred values of coupled variables versus time plots
- the initial window-size is 0.1
- the relative window-size limits are $0.5H_n$ and $2H_n$, where H_n is the size of the window just executed
- the absolute window-size limits H_{min} and H_{max} are 10^{-6} and 1 respectively.

For all the tests, if the next window ends past the simulation endpoint, the window-size is reduced so that the window ends exactly at the simulation endpoint. That is, if $T_{m+1} > S_{end}$ for the next window $[T_m, T_{m+1}]$, then $[T_m, T'_{m+1}]$ is used instead, where $T'_{m+1} = S_{end}$.

4.5 Testing Environment and Computer Programs for Testing and Plotting

The canned data used for testing, the programs for testing and plotting and the results are all available as a .zip file: *http://www.cs.toronto.edu/pub/reports/na/Rohan.MSc.Supp.2015.zip*. The .zip file contains a folder called *cdt* (which is an acronym for canned data testing). The sub-folders of the *cdt/* folder are

- cdt/canned_covars_data/
- cdt/results/
- cdt/plotting/
- cdt/CannedDataTesting/
- cdt/scripts/

Next, the the contents of each of the above sub-folders are explained. Note that all the source code authored by us and all the binaries provided by us are free software under the GNU General Public License (GPL) version 3.

4.5.1 cdt/canned_covars_data/

This folder contains the canned data for the coupled variables for the five test cases used for testing (*covar* is short for coupled variable). Specifically, for each test case, the values of each of the three coupled variables of the test system are stored in a single file (.csv) for $t \in [0, 3.5]$ with about 40 time points. The folder also contains *copies* of the aforementioned .csv files. Do not delete these copies. They are required by the program that executes the tests, which is discussed in §4.5.5. Finally, the folder contains a sub-folder called *canned_covars_plotted* in which there are five Microsoft Excel files, one for

each test case. Each of these Excel files contains the canned data for the corresponding test case as well as a plot of that data.

4.5.2 cdt/results/

This is the folder where the results of testing are automatically stored. There are three sub-folders: *first_order*, *second_order* and *third_order*, which contain the results of the $p_D = 1$, $p_D = 2$ and $p_D = 3$ controllers, respectively. Within each of these folders, there are sub-folders for each controller of the corresponding p_D. The folders of non-deadbeat controllers are named $H<p_D><p_A><p_F>$. The folders of deadbeat controllers, on the other hand, follow the same naming system except that $\_d$ is appended to the end of the name (so, $H<p_D><p_A><p_F>\_d$ is used instead). So, for example, within the folder *second_order*, there are sub-folders *H211*, *H220_d* and *H211_d*.

Within each non-deadbeat controller folder, there are sub-folders for each α, β parameter combination tested. For example, in the folder *H211*, there are sub-folders *b1_0.12500_b2_0.12500_a2_0.12500* (which corresponds to $H211b$ with $b = 8$), *b1_0.16667_b2_0.16667_a2_0.00000* (which corresponds to $H211\ PI$), among others. Within each of these particular non-deadbeat controller folders, there are two sub-folders, *CE* and *LE*, which contain the raw results (.csv files) and plots (MATLAB .fig and PDF files) for all the test cases for both the LD and LDIC methods using constant and linear extrapolation, respectively.

Since there is only one deadbeat controller for any combination of p_D, p_A and p_F, there is no intermediate layer of sub-folders for deadbeat controllers. That is, immediately within a deadbeat controller folder lie the *CE* and *LE* sub-folders.

4.5.3 cdt/plotting/

This folder contains the MATLAB programs required to plot the results of tests and save them (in MATLAB .fig and PDF formats) and a *Licenses.txt* (explained below). The file *CdtResultsPlot.m* is the main file. It makes use of the *export_fig* package of scripts and *suptitle.m*, which are free software released under a BSD-like License and the MIT License respectively. Both these licenses can be found in *Licenses.txt*. Note that the program *suptitle.m* has been modified. See the header of the file *CdtResultsPlot.m* for information on how to use it.

4.5.4 cdt/CannedDataTesting/

This folder is an Eclipse CDT C project. Its sources are in */src* (*main.c*, *CannedDataTesting.c* and *CannedDataTesting.h*) and its executable is in */Release*, named *CannedDataTesting.exe*. *CannedDataTesting.exe* executes a single test case for either the LD or the LDIC method with either CE or LE (these choices are made via the arguments). See the header of the file *main.c* for information on how to use the program. Note that GNU GCC was used as the compiler.

4.5.5 cdt/scripts/

This folder contains a number of Perl programs (for each program, see the header of the file for information on how to use it):

- *doController.pl* – for a single controller, which is identified using the arguments, this program executes the C program *CannedDataTesting.exe* for each test case, once for each combination (a, b), where $a \in \{\text{LD}, \text{LDIC}\}$ and $b \in \{\text{CE}, \text{LE}\}$.

- *doAllControllers.pl* – calls *doController.pl* for each and every controller, one after the other. The controller $H110$ is run five times, each time with a different β_1 value from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. The controllers $H211b$ and $H321b$ are run four times each, each time with a different b value from the sets $\{2, 4, 6, 8\}$ and $\{4, 8, 12, 16\}$ respectively.

- *compareResults.pl* – generates clustered bar graphs to compare the number of rejected windows and the total number of windows across controllers for every test case, once for every combination (a, b), where $a \in \{\text{LD}, \text{LDIC}\}$ and $b \in \{\text{CE}, \text{LE}\}$. One can choose, via the argument, to compare all the controllers, all the deadbeat controllers, all the non-deadbeat controllers, $H110$ controllers with $\beta_1 = 0.2, 0.4, 0.6, 0.8$ and 1.0, $H211b$ controllers with $b = 2, 4, 6$ and 8 or $H312b$ controllers with $b = 4, 8, 12$ and 16. When all the controllers are compared, $H110$ with $\beta_1 = 0.2$ is used, $H211b$ with $b = 8$ is used and finally, $H312b$ with $b = 12$ is used (the reasoning for this is given in §4.6.3.1, but, to understand it, you need to have read the text before §4.6.3.1 in §4.6.3). This program makes use of *bargraph.pl*, which is free software under the GNU GPL License, and indirectly uses *gnuplot* and the *transig* package, which are also free software, the licenses for which can be found in *Licenses.txt*.

- *compareResultsAcrossImpl.pl* – generates clustered bar graphs of the differences in the number of rejected windows and the total number of windows between two sets of results for a given set of controllers and for every test case, once for every combination (a, b), where $a \in \{\text{LD}, \text{LDIC}\}$ and $b \in \{\text{CE}, \text{LE}\}$. In addition to the graphs of the raw differences, it generates graphs of the percentage differences. Each set of results, i.e., the folders *first_order*, *second_order* and *third_order*, need be placed inside a folder. The names of the two folders should be supplied as the second and third program arguments (the graphs plot the change going from the first set of results to the second). The first argument specifies the group of controllers that the comparison should be done for. Refer to the program header for a list of valid controller group arguments (examples are "all" for all the controllers, "db" for the deadbeat controllers and "pDeq3" for the controllers with $p_D = 3$). This program can be used to determine the effect that a change in the implementation has in the results. For example, one could compare the results obtained using a absolute maximum H_{max} with the results obtained using a different absolute maximum H'_{max}. Finally, this program also makes use of *bargraph.pl*, *gnuplot* and the *transfig* package which, once again, are free software (see *compareResults.pl* paragraph above for licenses information). Note that if the number of rejected windows or the total number of windows of a test from the first result set is 0, then for the percentage difference for that test, we use the raw difference, since otherwise we would have to divide by zero. This does not happen though (for non-trivial problems) since the number of rejected windows and the total number of windows are always greater than 0.

4.6 Analysis of Results

Code A's coupled variables were "captured" successfully by Code B, i.e., the values Code B used for Code A's coupled variables were within the respective coupled variables' deviation tolerances, and vice versa, when the LDIC method was used (regardless of the controller, the test case or whether CE or LE was used). On the other hand, when the LD method was used, sometimes spikes were missed. An example of this is the narrow spike test for $H211\ PI$ with CE, the plots of which are shown in Figure 4.1. This is an expected drawback of the LD method. It is always *possible* that a spike will be missed when the LD method is used because the window could step over the spike or end near the end of the spike such that the window endpoint deviation is less than the tolerance. In contrast, the LDIC method will never miss a spike (at least for these simple tests) since it computes and checks the deviations for all the internal time-points and since, there must be at least one internal time-point if there exists a spike inside the window. This shortcoming of the LD method makes it *less reliable* than the LDIC method.

Note that the 2×2 plots (e.g., Figure 4.1) of all the controllers cannot be included in this document as there are too many plots (22 unique controllers were tested and there are 10 plots for each controller; thus, there are 220 2×2 plots). Some of these plots are included in this report to explain various points (e.g., Figure 4.1 illustrates how the LD method can miss spikes). Also, all the plots for the elementary controller, H_0110, can be found in Appendix B. As stated previously, the plots of *all* the controllers tested can be found in the folder *cdt/results/*. As noted earlier, the folder *cdt/* is contained in *http://www.cs.toronto.edu/pub/reports/na/Rohan.MSc.Supp.2015.zip*.

4.6.1 Comparison of LD and LDIC Results

The LD and LDIC results are either very similar or identical for all our tests, except for those where a spike is missed by the LD method, as discussed before. We explain why that is below.

First, let's consider CE. For all the tests, in almost all the windows, all the coupled variables are monotonic. For example, the spike and narrow spike cases have just one non-monotonic window (the window that encompasses the tip of the spike) and the sinusoids case has just six non-monotonic windows (the windows that encompass the peaks/troughs of the sinusoids).

For monotonic windows, meaning windows where all the coupled variables are monotonic, when CE is used, the window endpoint deviation will be larger than all interior point deviations for all the coupled variables. This is because the absolute difference between a monotonically varying coupled variable's value and its window starting point value grows monotonically over time, over the course of the window. Consequently, for LDIC and $p_D = 1$, the size of the window following a monotonic window is determined by the monotonic window's endpoint (and not by its interior points). As a result, the LD and LDIC methods produce the same result for the size of the next window, if the previous window's size is the same in both cases. For $p_D > 1$, the previous p_D windows' sizes and deviations need to be the same between LD and LDIC (the later will be true if the previous p_D windows are monotonic). So, for example, if all windows starting from the first to the m^{th} window are monotonic, then the LD and LDIC results will be identical in that period (assuming the initial window-sizes are the same for both tests, which is true for all our tests). If a non-monotonic window is encountered, the LD and LDIC

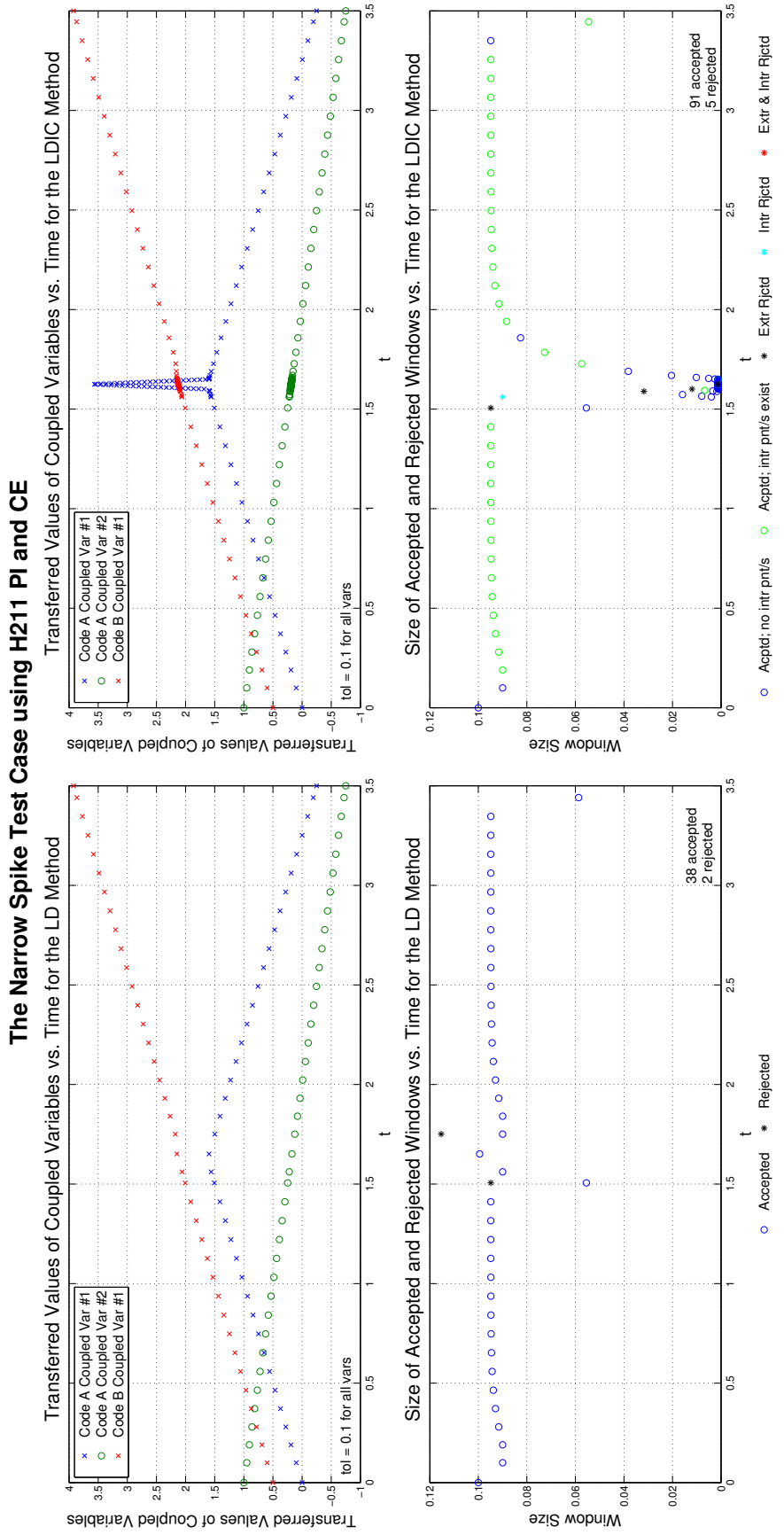

Figure 4.1: Results for the *H211 PI* controller with CE for the narrow spike test case. Note that the narrow spike is missed by the LD method but not by the LDIC method. It is always *possible* that the LD method will miss a spike but the LDIC method will never miss a spike (at least for these simple tests). This is a drawback of the LD method.

methods may still produce the same window-size, as explained below. If they don't produce the same window-size, the final window-sizes plot could still be very similar because the controllers are adaptive and there are only a few non-monotonic windows, at most.

The LD and LDIC methods *may* produce the same window-size for the next window even if the previous window was non-monotonic (assuming that, for $p_D > 1$, the $p_D - 1$ windows before the previous window are monotonic and that the window-sizes of the p_D previous windows are the same). For example, suppose that CE and LDIC are used and one of the coupled variables of one of the codes has just one interior point in some window. Suppose also that the value of the coupled variable at the interior time-point is greater than its window starting point value and that its window endpoint value is lower than its window starting point value by more than the amount by which its starting point value is lower than the interior point's value. Then, even though this window is non-monotonic, the endpoint deviation for the coupled variable is larger than the interior point's deviation and so, the next window's size will be determined by the window endpoint. As a result, the LD and LDIC methods will produce the same result for the next window, even though the previous window is non-monotonic.

Now, consider LE. For LE, unlike for CE, it is not necessarily true that the window endpoint deviation will be larger than all interior point deviations for monotonic windows: if two successive monotonic windows contain an inflection point, it is *possible*, with LE, for the window endpoint deviation of the second window to be smaller than the deviations of one or more internal time-points. However, three of the five test cases (spike, narrow spike and combination) don't have any inflection points! So, for these cases, the arguments above for CE apply directly. Now, consider the remaining two cases. The steep drop case has just one inflection point and the sinusoids case has only a few. Given that the number of inflection points is very small compared to the total number of accepted windows, the final window-sizes plot will be the same (the presence of an inflection point does not necessarily mean that the window endpoint deviation will not be the largest) or be very similar (the controllers are adaptive).

4.6.2 Comparison of CE and LE Results

4.6.2.1 Window-Size Sequences

It is observed in the results that in regions where all the coupled variables are varying linearly, for both the LD and LDIC methods,

- if LE is used, the window-size continually grows until it becomes as large as the absolute window-size limit (e.g., see regions $t = [0, 1.5]$ and $t = [1.8, 3.5]$ in Figure 4.2). This behaviour is expected, since, when linear extrapolation is used and all the coupled variables are varying linearly, the deviation for every coupled variable will be zero. So, the window-size will, at every step, be increased by the relative upper limit, until it reaches the absolute upper limit.
- if CE is used, the window-size remains constant, after an initial rise or decline to the constant value. In a region where the coupled variables are all varying linearly, the window-size is limited by the steepest slope. This is because the deviation varies linearly and positively with the window-size when CE is used. When the coupled variables enter a linear region, the window-size grows or shrinks, depending on the starting window-size, until that steepest-slope-determined largest

possible window-size is reached. Then, the window-size remains constant until the linear region ends. For an example, see Figure 4.1.

It is observed in the results that in regions where all coupled variables are changing *slowly*, for both the LD and LDIC methods,

- if LE is used, the window-size grows until the deviation becomes larger than tolerated or the window-size computed for the next window is larger than the absolute upper limit. For an example, see the *bottom* plot of Figure 4.8. This is expected, since, over sufficiently short windows, the linear-extrapolation-determined deviation of a slowly varying coupled variable will be smaller than the tolerance and so, the window-size, at every step, will be increased (the increase may be limited by the relative upper limit of course). However, eventually one of two things will happen, depending on which one comes first. First, the deviation could grow to be larger than the tolerance (over a sufficiently large window, the slowly changing coupled variable will have a large deviation) and so, the window-size will be reduced. Second, the window-size could be constrained by the absolute upper limit.

- if CE is used, the window-size changes gradually. Over sufficiently short windows, the variables will be varying close to linearly. Then, the window cannot exceed a certain size that is determined by the steepest slope among the coupled variables for that window. This slope slowly changes as time progresses since the coupled variables are changing slowly. So, the window-size also slowly changes.

4.6.2.2 Number of Accepted and Rejected Windows

For our test cases, it is observed that, when LE is used, as opposed to CE, the number of accepted windows is significantly smaller. For most of the simulation time period, in all the test cases, all coupled variables vary linearly or close to linearly, even over fairly large windows. As a result, the deviation/s for these windows are significantly smaller when LE is used as opposed to CE. Consequently, significantly larger window-sizes are used for LE compared to CE (these sizes may be limited by the relative or absolute limits), which in turn means that there are significantly fewer accepted windows for LE than for CE.

On the other hand, when LE is used, there are many more rejected windows for most of the test cases with most of the controllers. One can see this visually by comparing the top plot (CE) and bottom plot (LE) of Figure 4.4. When LE is used and when a kink is encountered during the simulation (such as at the start of a spike, at the peak of a spike or at the start of a steep drop), the window-size needs to decrease much more for LE than for CE in order for the next window to be accepted, because the window-size starts off at a much larger value for LE compared to CE. If the window-size reduction required is large, the controller logic cannot achieve the required decrease in one step but rather, it requires many attempts. Each failed attempt reduces the window-size further until a sufficiently small window-size is reached. So, there are more rejected windows when LE is used, especially when the simulation has many kinks (for example, the combination case has more kinks than the narrow spike case).

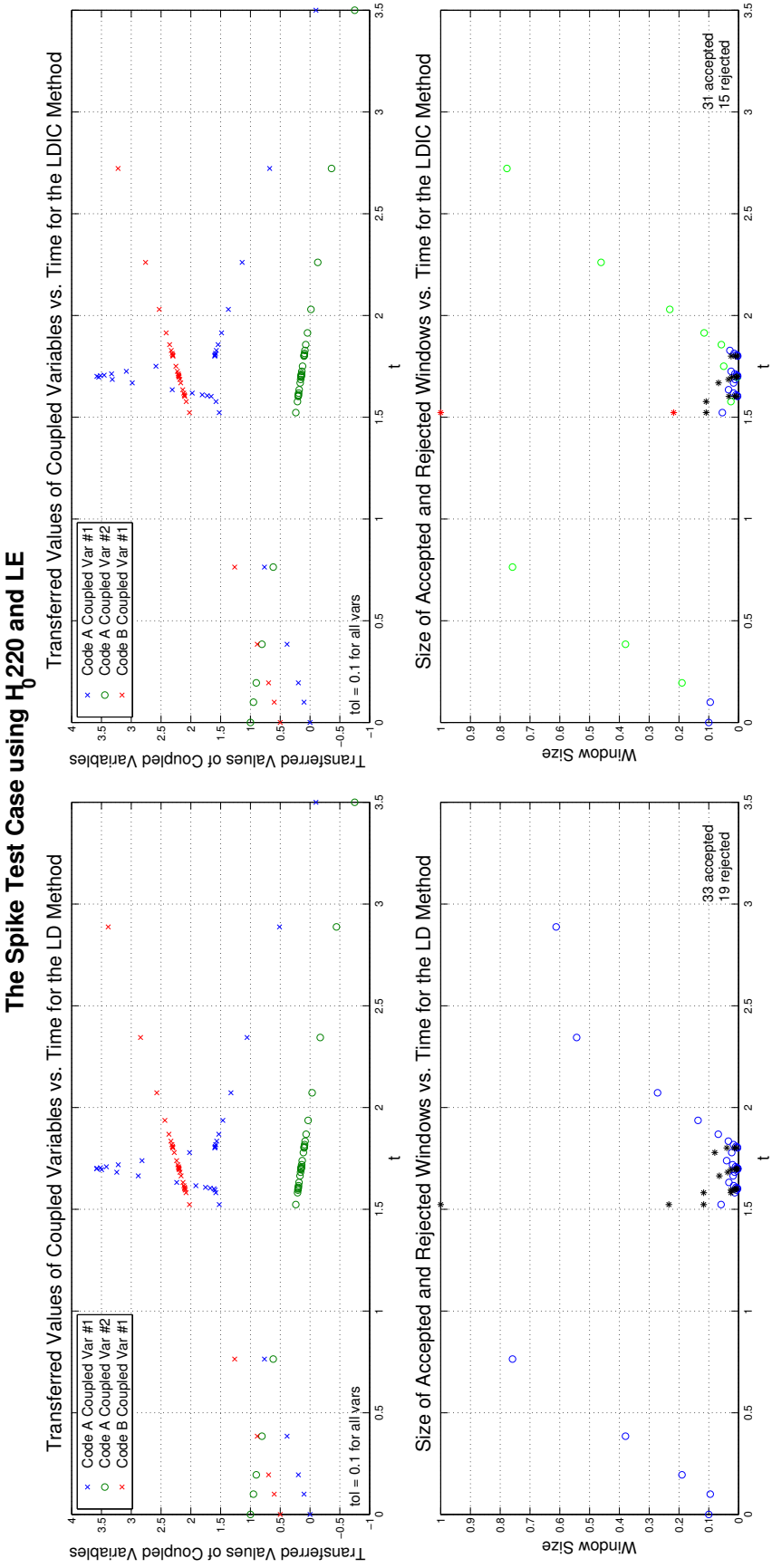

Figure 4.2: Results for the H_0220 controller with LE for the spike test case. Note that the window-size grows at every step in the linearly varying regions.

However, the decrease in the number of accepted windows going from CE to LE is significantly greater than the increase in the number of rejected windows. So, for all our test cases and for all the controllers, the total number of windows is significantly smaller when LE is used. This can be seen readily from the bar graphs in Figure 4.3 (left corresponds to LDIC with CE and right to LDIC with LE). Note that this subsection applies to both the LD and LDIC methods.

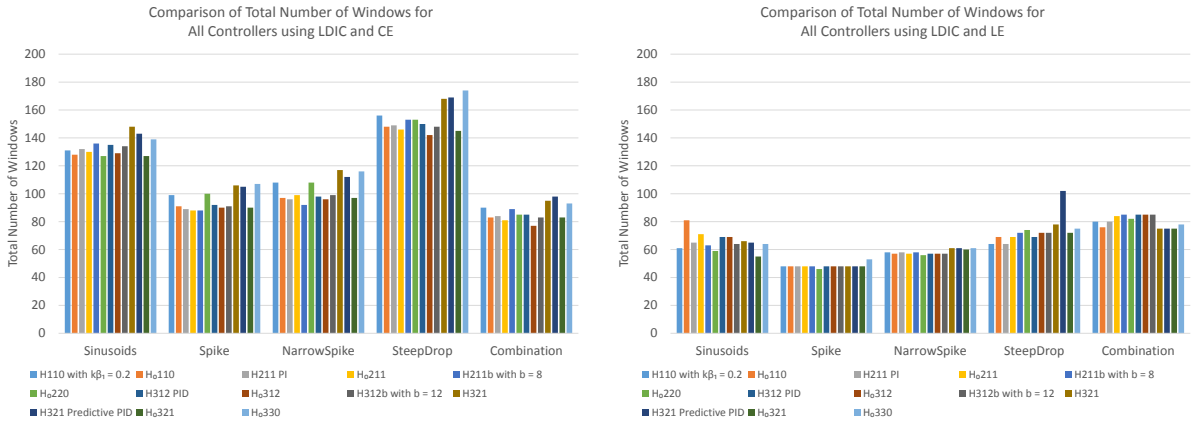

Figure 4.3: The total number of windows for all the test cases and all the controllers. The left graph corresponds to LDIC with CE and the right to LDIC with LE. Note that across all the controllers and all the test cases, the total number of windows for CE is significantly larger than that for LE.

4.6.3 Comparison of Controllers

The total number of windows, i.e., the number of accepted windows plus the number of rejected windows, determines the amount of data transferred: the greater the number of windows, the greater the number of values transferred.

Computational work increases when the number of accepted windows for the simulation increases because of the two following reasons (assuming that the internal time-steps used by the codes are about the same for the windows):

1. each code will have to use linear interpolation (or some higher-order interpolation) to determine the values of its coupled variables at the window endpoint of each extra window. This is because, most likely, the last internal step will step over the window endpoint. Note that while it is possible for each code to compute the values of its coupled variables at the window endpoint by altering the last internal step's size so that it ends at the window endpoint, that is an inferior choice to using interpolation. There are two reasons for this. First, the cost for a code to compute the window endpoint values using it's solver (e.g., solver for a system of PDEs) is normally much higher than that of interpolation. Second, the internal time-stepping process would be interrupted by altering the size of some of the time-steps, i.e., the last time-step of each window.

2. the controller formula (along with associated logic) will have to be executed for each extra window to determine the next window's size.

   On the other hand, computational work increases when the number of *rejected* windows for the

simulation increases because of the two reasons listed above *as well as* a third additional reason: for each extra rejected window, each code will have to compute the values of its coupled variables at all its internal time-points and the window endpoint, assuming that it did not save the values at its internal time-points from the previous (rejected) window. The internal time-points for the new window, i.e., after the rejection, and the values at those time-points, will be the same as those of the rejected window since the initial conditions for both windows are the same! So, if, throughout the simulation, the values at the internal time-points of the most recent window are saved by each code, then, additional rejected windows are not more expensive then additional accepted windows. We were not aware of this alternate implementation until late in our work and so, in this document, we assume the codes do not store the values in memory.

Now, assume that

1. the sizes of the windows is normally much greater than the sizes of the internal time-steps. That is, assume that all windows contain many internal time-steps.

2. most of the computational work is done by the codes, i.e., the controller and interpolation costs are small compared to the cost of the codes' solvers. This assumption is reasonable for complex codes that solve large non-linear systems of PDEs, such as those of AECL.

3. the time required for data to be transferred between codes, for the controller overhead and for interpolation is small compared to the time required for all the codes to execute a window. This assumption is reasonable for complex codes, such as those of AECL.

If assumptions 1-2 are true, then, the increase in computational work due to an increase in the number of accepted windows is small compared to the increase in computational work due to rejected windows. If assumption 3 is also true, then the same is true for computing time, not just computational work. Thus, the difference in the overall computational work or computing time, between two controllers, is determined primarily by the difference in the number of rejected windows and not by the difference in the total number of windows. In the rest of the document, we use the term "performance" in the context of comparing controllers to mean the following: better performance means smaller overall computational work or equivalently, shorter computing time and poorer performance means greater overall computational work or equivalently, longer computing time. So, using the term performance, the sentence two sentences back becomes: the difference in the performance between two controllers is determined primarily by the difference in the number of rejected windows and not by the difference in the total number of windows.

One may think that using very small windows is a good choice since that will drastically reduce the number of rejected windows and, thereby, minimize the computational work or equivalently, the overall computing time. However, that is not so. The smaller the windows (equivalently, the greater the number of windows), the greater the controller and interpolation costs, due to the reasons given earlier in this section. If the number of windows is sufficiently large, the controller and interpolation costs will become significant compared to (or possibly even exceed) the cost of the codes. Further, the greater the number of windows, the greater the time required for data transfer and, so, if the number of windows is sufficiently large, the data transfer time will also become significant compared to (or possibly even exceed) the time taken by the codes to execute all the windows.

Some windows may have more internal time-steps than others. Indeed, some windows may have significantly more internal time-steps. So, some rejected windows may be more expensive than others. So, one might ask whether a particular controller or set of controllers has fewer rejected windows in regions where the density of internal time-steps is high. While for a particular system and test case, one controller or some controllers may do better than others in this respect, we believe they cannot do so in general. This is because controllers have no way of predicting whether an upcoming window will require many internal time-steps for one or more codes. So, if a controller has fewer rejected windows relative to other controllers in regions with high-density of internal time-steps, we believe that is merely due to chance.

The extra computational work done due to rejected windows is roughly $\frac{x}{y}W$, where x is the total number of internal time-points of all the rejected windows, y is the total number of internal time-points of all the accepted windows and W is the work that would be done if there were no rejected windows. This of course assumes that the work required to compute values for all internal time-points is the same. That is reasonable, since the same solver will be used by a code across all its internal time-points and the work done by a solver is usually about the same for all internal time-points.

For the reasons explained above, we believe that the best window-size controllers are those that have the fewest number of rejected windows, assuming that they don't have an unreasonably large total number of windows.

4.6.3.1 Comparing the Number of Rejected Windows of Controllers

As mentioned previously, all controllers were tested using the five test cases. For each controller, each test case was run once for each combination (a, b), where $a \in \{\text{LD}, \text{LDIC}\}$ and $b \in \{\text{CE}, \text{LE}\}$.

The two graphs in Figure 4.4 are clustered bar graphs that allow for easy comparison of the number of rejected windows of the various controllers for each test case; the top graph corresponds to the LDIC method with CE and the bottom one, to the LDIC method with LE. As stated in the legend, for the controllers $H110$, $H211b$ and $H312b$, we use $\beta_1 = 0.2$, $b = 8$ and $b = 12$ respectively. These values were determined via testing to be the best, or at least as good as any others, in terms of the number of rejected windows for the respective controllers. The total number of windows for these parameter values was not much larger than that of the other values.

When CE is used, the $H321$ controller is the best choice for our test cases. Refer to the top graph in Figure 4.4 for the following discussion. In both the sinusoids and the steep drop test cases, it is the best and does significantly better than most other controllers. In the spike and narrow spike test cases, it ties for best and is just a little worse than the best, respectively. Note however that the differences in the number of rejected windows for these two cases aren't significant for many of the controllers as explained at the end of this section. Finally, in the combination case, it is just a little worse than the best. So, overall, $H321$ is definitely the best choice for CE for our test cases.

When LE is used, on the other hand, no one controller stands out as the best choice for our tests, but there are a few that perform significantly worse than the others for the sinusoids test case. Refer to the bottom graph in Figure 4.4 for the following discussion. For the sinusoids case, H_0110, H_0211

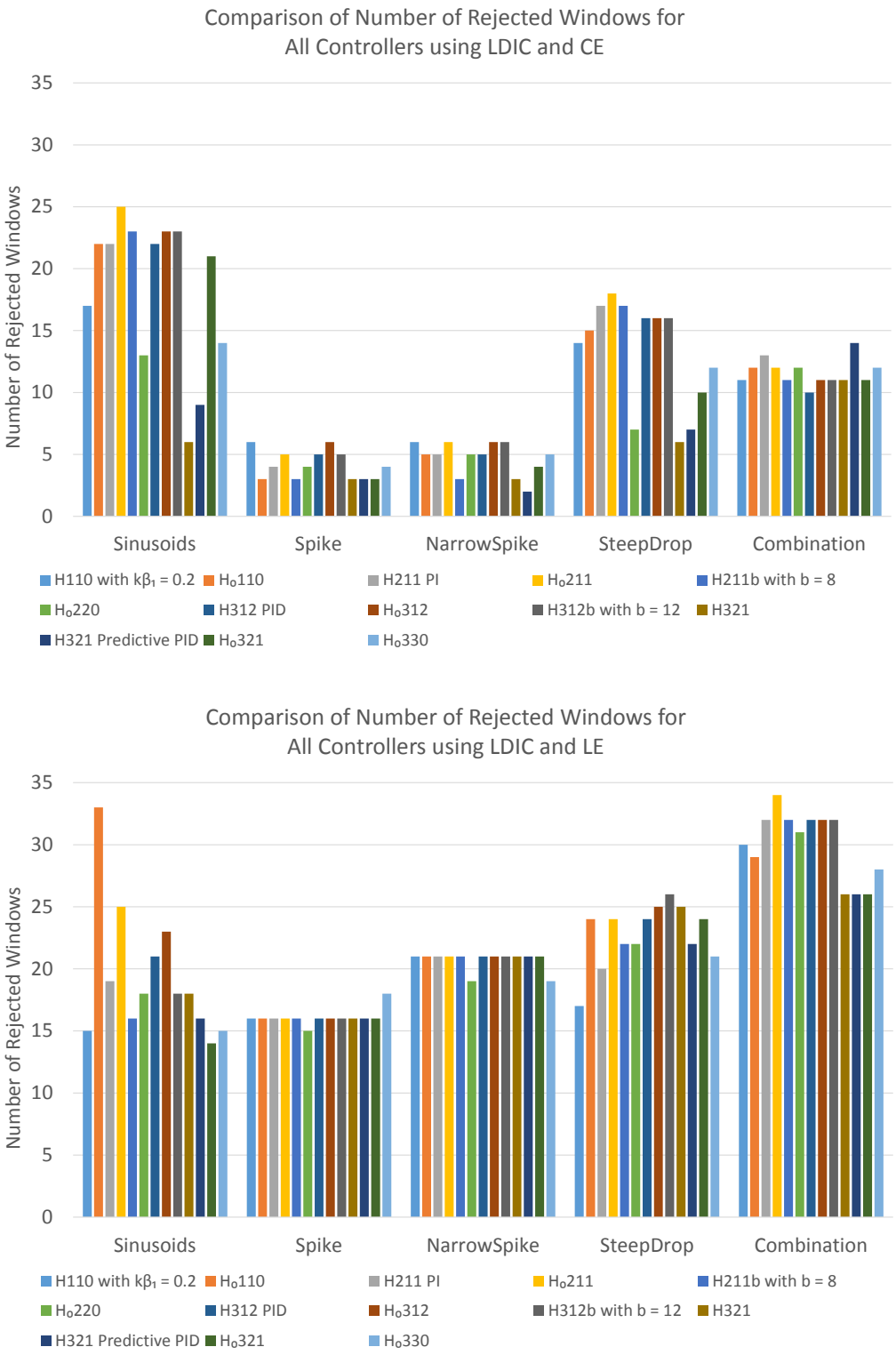

Figure 4.4: The number of rejected windows of all the controllers are compared against each other for each test case, for the LDIC method with CE (top) and for the LDIC method with LE (bottom). Note that across all the controllers and all the test cases, the number of rejected windows for LE is greater than that for CE.

and $H312\ PID$ are significantly worse than most of the others. The remainder are not significantly different from one another. So, based on our test cases, when LE is used, the controllers H_0110, H_0211 and $H312\ PID$ should be avoided and any one of the remaining controllers could be used as they all provide roughly the same performance.

Note that, for most cases and most controllers, the number of rejected windows when LE is used is significantly larger than that when CE is used. This is expected as explained in §4.6.2.2. In addition, note that we do not perform the number-of-rejected-windows comparison for the LD method also because the LD method is unreliable in some cases, as explained in the beginning of 4.6, and because, for the remaining cases, the LD method's results are similar to or identical to those of the LDIC method for *our* test cases, as explained in 4.6.1.

Suppose that, for the same test case, e.g., combination or steep drop, two tests are carried out with a difference in the choice of γ and/or the choice of controller and/or the choices for one or more of the other parameters. Then, a difference of a few rejected windows between the two tests is not necessarily due to a difference in the choices referenced above. It may instead be due to chance. We illustrate this with the following example. Suppose that

1. CE and LDIC are used
2. there exists a coupled variable x in code c
3. for code c, the window $[T_m, T_{m+1}]$ has one and only one internal time-point t_p where $T_m < t_p < T_{m+1}$
4. $x(t_p) > x(T_m)$ and $x(T_m) = x(T_{m+1})$
5. $dev_x^{t_p} =\mid x(t_p) - x(T_m) \mid > tol_x$

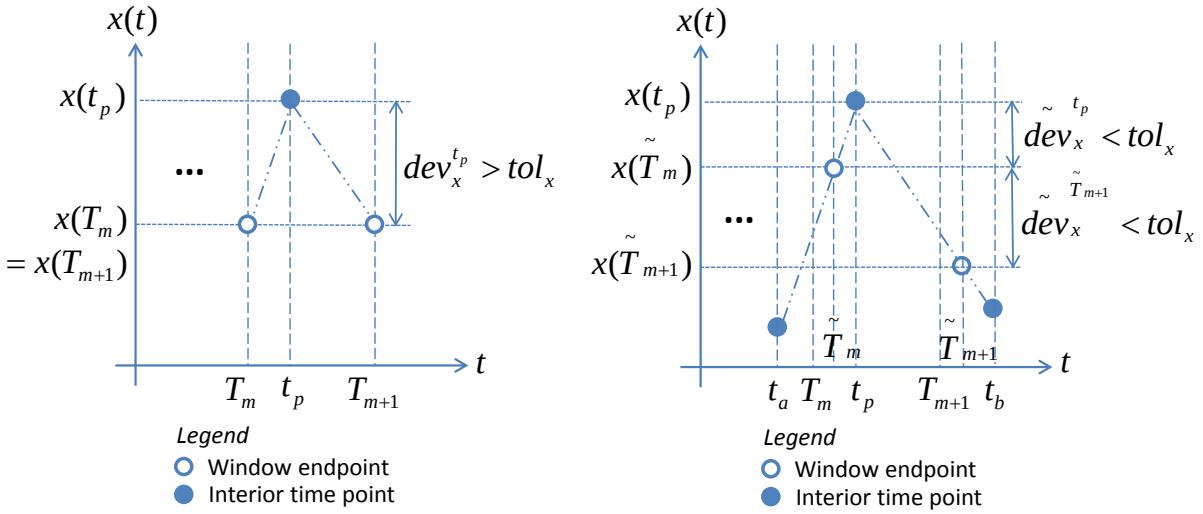

Figure 4.5: This figure illustrates that a difference of a few rejected windows between two runs of the same test case, e.g., combination or steep drop, with a difference in the choice of controller, γ, etc. could be due to chance and chance alone, i.e., not due to the difference(s) in the choice of controller, γ, etc.

The scenario described above is illustrated in the left plot of Figure 4.5. Since $dev_x^{t_p} =\mid x(t_p) -$

$x(T_m) \mid > tol_x$, the window will be rejected. Now, consider a second scenario: suppose that the window from the scenario described above started at the midpoint between T_m and t_p, instead of T_m, but that the window-size remains the same. Let $\tilde{T}_m = T_m + \frac{t_p - T_m}{2}$. Then, the new window is $[\tilde{T}_m, \tilde{T}_{m+1}]$, where $\tilde{T}_{m+1} = T_{m+1} + \frac{t_p - T_m}{2}$. Assume that there exists a canned data point at $t_a < T_m$, which is true for all windows except the first, and that t_a is the closest canned data point to T_m that is also less than T_m. Then, $x(\tilde{T}_m) = x(t_a) + \frac{x(t_p) - x(t_a)}{t_p - t_a}(\tilde{T}_m - t_a)$ since linear interpolation is used to compute values between canned data points. Suppose that $\widetilde{dev}_x^{t_p} = \mid x(t_p) - x(\tilde{T}_m) \mid < tol_x$ and $\widetilde{dev}_x^{\tilde{T}_{m+1}} = \mid x(\tilde{T}_{m+1}) - x(\tilde{T}_m) \mid < tol_x$ (while $\mid dev_x^{t_p} = x(t_p) - x(T_m) \mid > tol_x$ is still true). Then, the window will pass! This second scenario is illustrated in the right plot of Figure 4.5. So, whether or not there is a rejected window here is determined simply by the starting point of the window, i.e., chance. In general, the start and end points of windows in a region that has a peak or trough can alter the number of rejected windows by at least one for this reason. This situation applies to all the test cases since they all have at least one concave or convex region.

4.6.3.2 Inability of Some Controllers to Gradually Change Window-Size Without Any Rejections

It is observed in the results that many controllers successfully capture slowly varying coupled variables with some rejected windows, while others do so without *any* rejected windows. The latter is of course the preferred behaviour. For example, see the window-size versus time plots of $H211\ PI$ and H_0321 using LDIC and CE for the steep drop test case in Figure 4.6 ($H211\ PI$ is in the middle and H_0321 is on the bottom). The corresponding transferred values versus time plot of $H211\ PI$ is at the top of Figure 4.6. Consider the $t \in [2.5, 3.5]$ portion of the window-size versus time plots. One can readily observe that the slow decrease over time of the window-size in this region occurs with no rejected windows in the case of H_0321, whereas the decrease occurs with rejected windows in the case of $H211\ PI$ (the asterisk marks indicate rejected windows).

4.6.3.3 Undesired Behaviour in Some $p_D = 2$ and $p_D = 3$ Controllers

The larger the value of p_D, the less intuitive the behaviour of the controllers. It is possible for controllers with high order-of-dynamics to exhibit undesirable and seemingly strange behaviour. We observed two types of such behaviour in our numerical results.

The first type is a dip in the window-size in a region where a monotonic increase in the window-size is expected. This is observed for the narrow spike test case for one $p_D = 2$ controller and a few $p_D = 3$ controllers. See Figure 4.7 for the plots of one of these tests: the narrow spike test case using $H321$ with LDIC and CE. Refer to this figure for the following discussion. Due to the narrow spike, the window-size is reduced significantly. Following the narrow spike, the window-size should increase steadily, since all the coupled variables vary slowly (compared to the spike), as can be seen in the top plot in Figure 4.7. $H321$ does increase the window-size, as expected, but the increase is not monotonic or even close to being monotonic. Instead, the window-size increases after the spike, then decreases, then increases again. The reason that a monotonic increase is preferred is that the dip makes the average window-size smaller and as a result, the controller overhead and the data transfer time is larger.

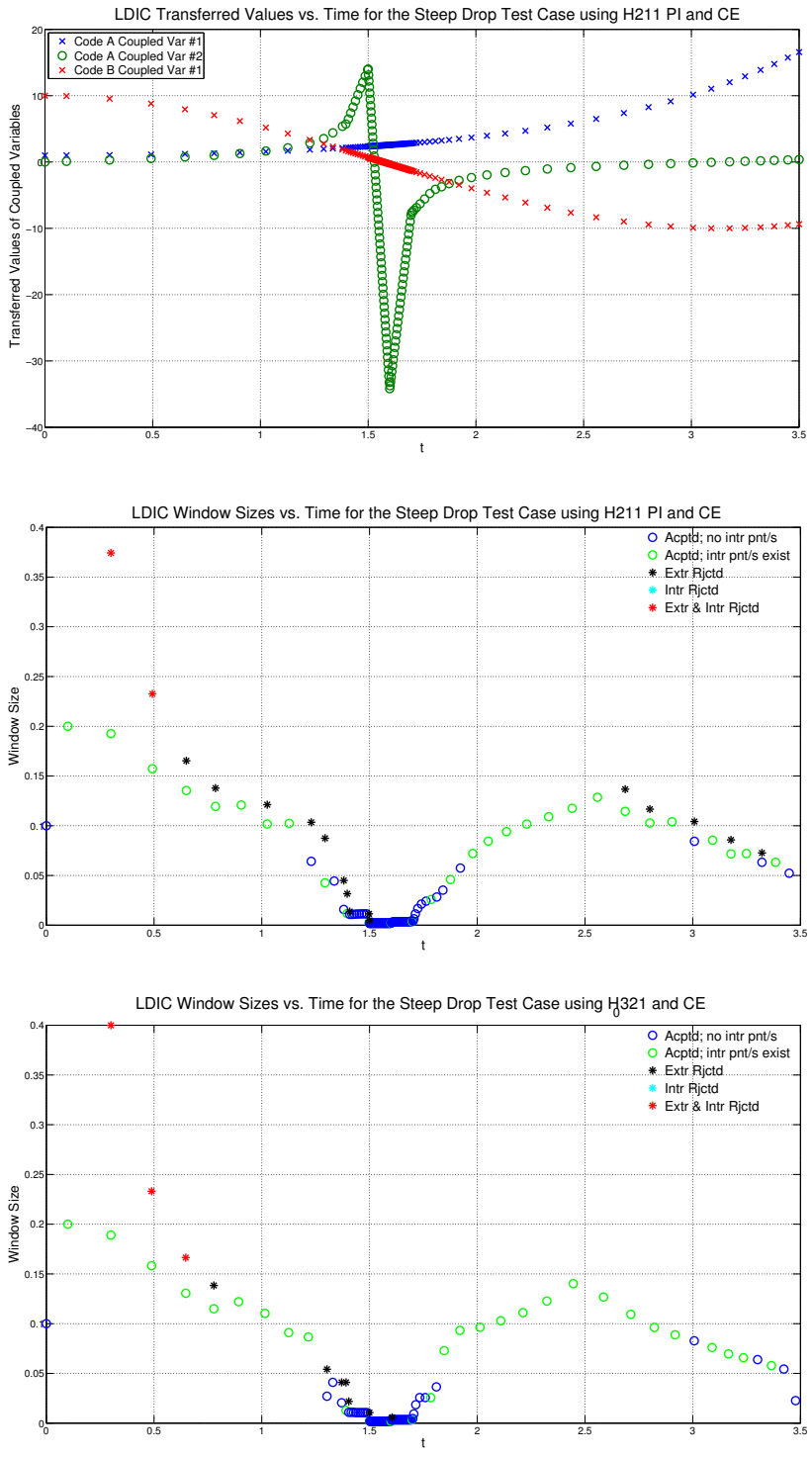

Figure 4.6: The transferred values versus time and window-size versus time plots of $H211\ PI$ with LDIC and CE for the steep drop test case are the top and middle sub-figures respectively. The window-size versus time plot of H_0321 with LDIC and CE for the steep drop test case is the bottom sub-figure. Notice that the necessary decrease in the window-size in the region $t \in [2.5, 3.5]$ is accomplished successfully by both controllers. However, H_0321 does so without any rejected windows, whereas $H211\ PI$ does so with five rejected windows (the asterisk marks indicate rejected windows).

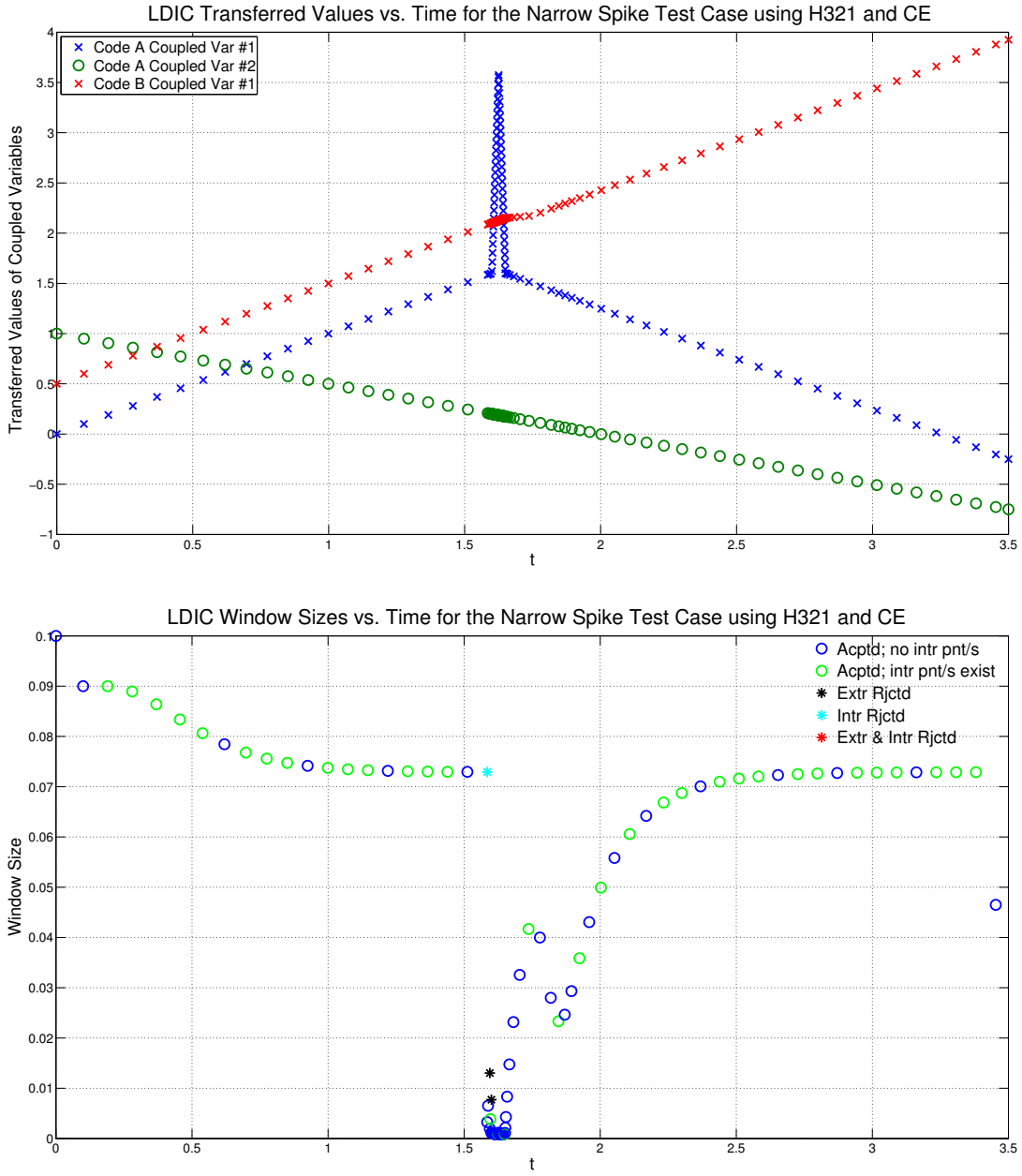

Figure 4.7: The plots for the narrow spike test case for $H321$ using LDIC and CE. Note that there is a dip in the window-size following the spike. It is possible and preferred for the window-size increase to be monotonic after the spike. Some controllers exhibit the undesired behaviour seen above, while most do not.

The second type of undesired behaviour observed in the results is oscillatory behaviour in the window-size sequence. This is observed for two tests only: the first is the steep drop test for the $H321$ *PID Predictive* controller with LE and LDIC and the second is the same except with LD instead of LDIC. In a region where one would expect an increase followed by a decrease in the window-size, the window-size in these cases oscillates! Of the other controllers, two have two or three oscillations and then recover and the remainder do not exhibit oscillatory behaviour at all. Refer to Figure 4.8 for the following discussion. The transferred values versus time and window-size versus time plots for the LDIC pathological case are shown in the top and middle plots respectively of Figure 4.8. Also included in the figure in the bottom plot is the window-size versus time plot for the $H312$ PID controller (also with LE and LDIC) for which there are no oscillations. Note that in the case of the $H321$ PI *Predictive* controller, in the region $t \in [1.7, 3.5]$, the oscillations continue all the way to the end of the simulation, whereas, in the case of the $H312$ PID controller, there is one small oscillation in the beginning of this region and then there is a monotonic increase followed by a monotonic decrease, as expected. Note that oscillatory behaviour in these pathological cases makes the average window-size significantly smaller and as a result, the controller overhead and the data transfer time become larger.

4.6.4 Effect of the Safety Factor in the Performance of Controllers

4.6.4.1 Explaining Safety Factor Effect for CE

The top two bar graphs in Figure 4.9 show the raw change and percentage change in the number of rejected windows when γ is changed from $\gamma = 0.9$ to $\gamma = 0.7$ for all the controllers with LDIC and CE. The bottom two bar graphs in Figure 4.9 show the same for the *total* number of windows.

The number of rejected windows is smaller for $\gamma = 0.7$ than for $\gamma = 0.9$ in most cases. This is expected since every window-size is reduced if γ is smaller and consequently, the number of rejections may be reduced. Note that, for the spike and narrow spike cases, if the number of rejected windows is only a few, it cannot be reduced further to 0 since a few rejections may be unavoidable as explained in §4.6.3.1.

One can see that the total number of windows increases in all cases, but the increase is much larger for a few controllers than for the others and, for one (H_0220), it is somewhere in between. Also, the increase in the total number of windows in these particular controllers is smaller in the combination test case than in the other test cases. Next, we explain these observations.

First, we derive the approximate relationship between window-size and gamma for the $n+1^{st}$ window assuming that in the previous p_D accepted windows, all the coupled variables are linear or approximately linear and that all coupled variables are monotonic (needed only for LDIC, not LD). Second, we show that, for all the test cases, there are many windows for which these assumptions hold and that, in fact, together, they constitute the majority of the domain. Finally, given that the derived approximate relationship applies to our tests, we explain how it explains the observed behaviour for the total number of windows.

For CE, as shown in §A.2,

$$dev_n = c_n H_n \tag{4.1}$$

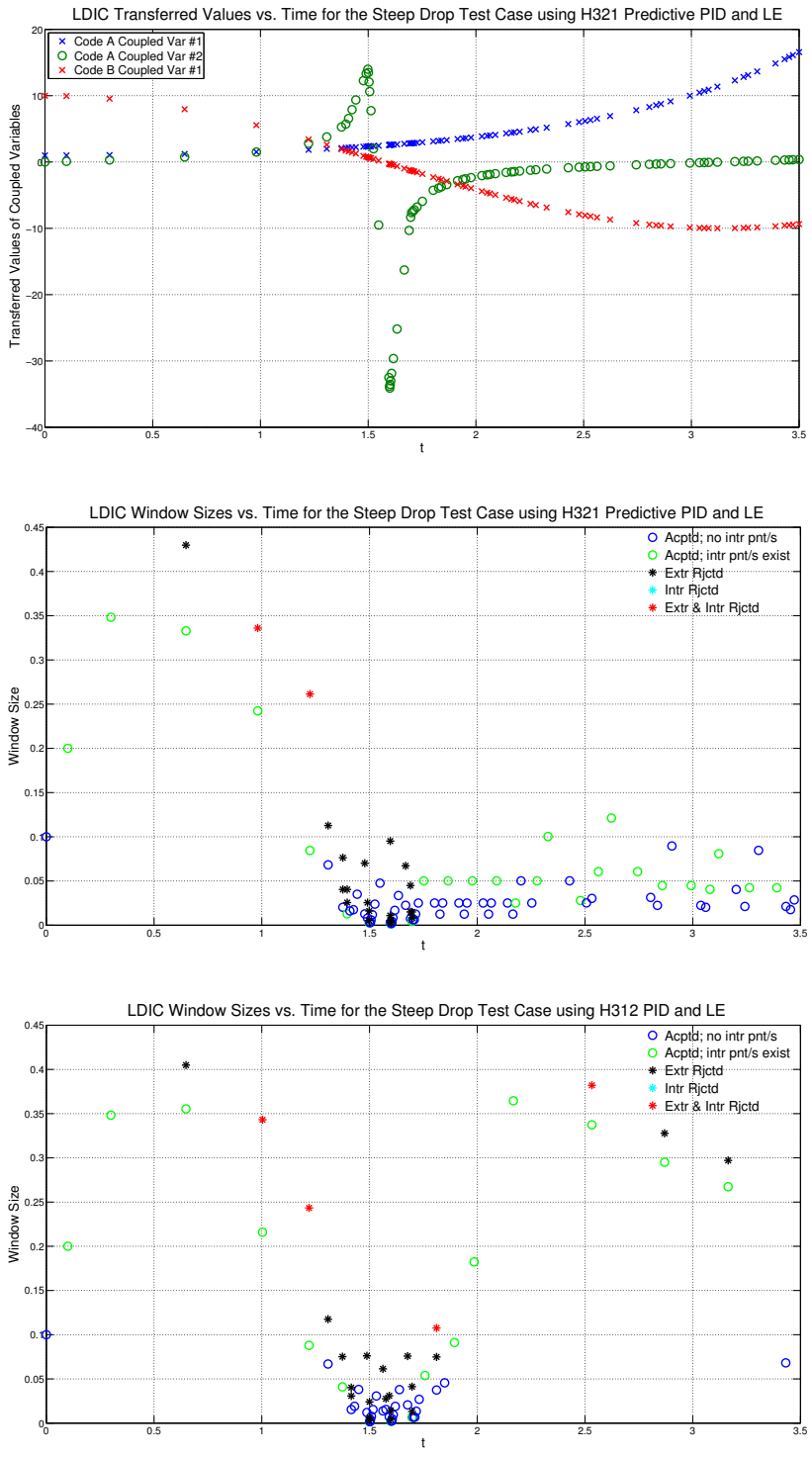

Figure 4.8: The window-size oscillates after about $t = 1.7$ for the $H321$ *PID Predictive* controller for the steep drop test case with LDIC and LE. This can be seen easily in the corresponding window-size versus time plot (middle). It is also observable in the corresponding transferred values plot (top) by the spacing in time of the transfers. The corresponding window-size versus time plot of the $H312$ PID controller is included in the bottom plot of the figure. $H312$ PID does *not* exhibit oscillatory behaviour in the window-size.

where H_n is the size of some window $[T_m, T_{m+1}]$ and $c_n = x'(\theta_t)$ for some $\theta_t \in [T_m, T_{m+1}]$.

If $c_n \approx c$, for some constant c, over several windows, then, in that region,

$$dev_n \approx cH_n \tag{4.2}$$

Using (4.2) and the fact that the controller tries to make $dev_n \approx tol$,

$$H_n \approx H^* \tag{4.3}$$

Then, using (4.2), (4.3) and (3.8), the equation of the general third order controller for the LD method (considering just a single coupled variable),

$$\begin{aligned} H^* &= \left(\frac{\gamma \cdot tol}{cH^*}\right)^{\beta_1} \left(\frac{tol}{cH^*}\right)^{\beta_2} \left(\frac{tol}{cH^*}\right)^{\beta_3} H^* \\ &= \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} \frac{H^*}{H^{*\beta_1+\beta_2+\beta_3}} \\ &= \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} H^{*1-(\beta_1+\beta_2+\beta_3)} \end{aligned} \tag{4.4}$$

$$H^{*\beta_1+\beta_2+\beta_3} = \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} \tag{4.5}$$

Therefore,

$$H^* = \gamma^{\frac{\beta_1}{\beta_1+\beta_2+\beta_3}} \left(\frac{tol}{c}\right) \tag{4.6}$$

A few things to note:

1. although we used the general third-order controller formula for the LD method (3.8) above, this applies to all controllers: for second-order controllers, simply set $\beta_3 = 0$, and for first-order controllers, simply set $\beta_2 = 0$ and $\beta_3 = 0$.

2. in the derivation, we consider one coupled variable only, but, if all the coupled variables in the system were considered, the relationship between window-size and γ would be the same. Since the window-sizes are small compared to the variations of the coupled variables (the deviations are limited by the LD/LDIC method), for the majority of the domain, over a stretch of $p_D \leq 3$ windows, the same coupled variable will normally have the largest deviation. This coupled variable will determine the next window's size as per (3.8), and so, if its c_n is roughly constant over the previous p_D windows, the same relationship, i.e., (4.6), will hold.

3. we did not consider internal time-points in the derivation. However, including them will produce the same result, i.e., (4.6), *if* the window endpoint deviation is greater than all interior time-point deviations for the previous p_D windows, which in turn will be true if all the coupled variables were monotonic in those windows.

So, if (4.2) is true for the previous p_D windows, then (4.6) follows. Eq. (4.2) is true for the previous p_D windows if, for the LD method, all the coupled variables are linear or approximately linear in those windows and, for the LDIC method, the same is true, and in addition, all coupled variables are monotonic in those windows.

Next, we consider each of the test cases to show that they meet the above conditions for the majority of the time domain:

1. for the spike, narrow spike and combination test cases, all coupled variables vary exactly linearly for the entire time domain, except for a few kinks (three for the spike and the narrow spike test cases and seven for the combination test case),

2. for the sinusoids case, one can see visually from the transferred values versus time and the window-sizes versus time plots (see the plots for $H110_0$ in Appendix B and the plots of the other controllers available in http://www.cs.toronto.edu/pub/reports/na/Rohan.MSc.Supp.2015.zip), that, for the majority of the domain, all couple variables are approximately linear and monotonic,

3. for the steep drop test case, for the vast majority of the domain, all coupled variables are exactly linear or approximately linear and monotonic (refer to the steep drop test case plots for confirmation).

Given that (4.6) applies to the majority of the domain for all the test cases, the smaller the value of H^* in (4.6), the greater the number of accepted windows, which in turn means, the greater the *total* number of windows (the number of rejected windows is usually smaller when a smaller γ is used, as explained above, but the increase in the number of accepted windows is greater than the decrease in the number of rejected windows).

From (4.6) we get,

$$\frac{H^*_{\gamma_2}}{H^*_{\gamma_1}} = \left(\frac{\gamma_2}{\gamma_1}\right)^{\frac{\beta_1}{\beta_1+\beta_2+\beta_3}} \tag{4.7}$$

where

$H^*_{\gamma_1}$ is the window-size of the $n+1^{st}$ window when γ_1 is used

$H^*_{\gamma_2}$ is the window-size of the $n+1^{st}$ window when γ_2 is used

For about half of the controllers tested, all β values are positive, and, for the others, one and only one of β_2 and β_3 is negative (β_1 is positive for all controllers; see Table 3.1 and Table 3.2). First, consider the controllers for which all β values are positive. For these, the exponent in the right side of (4.7) is positive and less than 1. Consequently, for $\gamma_2 < \gamma_1$, $H^*_{\gamma_2} < H^*_{\gamma_1}$, which in turn means that the total number of windows is larger when γ_2 is used than when γ_1 is used, as explained previously. This is exactly what we observe in Figure 4.9 for controllers with all positive β values: all these controllers have a positive change in the total number of windows going from $\gamma = 0.9$ to $\gamma = 0.7$.

Next, for the controllers with a negative β_2 or β_3, the exponent in the right side of (4.7) is greater than 1 for the controllers we tested, and consequently, for $\gamma_2 < \gamma_1$, $H^*_{\gamma_2} < H^*_{\gamma_1}$. While this is also true for the case where all β values are positive, there is a difference: $\left(\frac{\gamma_2}{\gamma_1}\right)^{\frac{\beta_1}{\beta_1+\beta_2+\beta_3}}$ here, because, in (4.7),

$\frac{\gamma_2}{\gamma_1} < 1$, and because $\frac{\beta_1}{\beta_1+\beta_2+\beta_3} < 1$ in the case where all β values are positive and $\frac{\beta_1}{\beta_1+\beta_2+\beta_3} > 1$ in the case where one β value is negative. Of course, for the controllers with one negative β, the degree to which $H^*_{\gamma_2}$ is smaller than $H^*_{\gamma_1}$ depends on the relative size of $\frac{\beta_1}{\beta_1+\beta_2+\beta_3}$, i.e. the larger $\frac{\beta_1}{\beta_1+\beta_2+\beta_3}$, the smaller $\frac{H^*_{\gamma_2}}{H^*_{\gamma_1}}$. This is exactly what we observe in the bottom plots of Figure 4.9 for the controllers with one negative β (remember that the smaller H^* is, the greater the *total* number of windows, as explained previously): all these controllers have a positive change in the total number of windows going from $\gamma = 0.9$ to $\gamma = 0.7$, the positive changes in the total number of windows for these controllers is much greater than that of the controllers with all positive β values, and finally, the relative sizes of the increase in the total number of windows among the one-negative-β controllers reflects the relative sizes of $v := \frac{\beta_1}{\beta_1+\beta_2+\beta_3}$: $H220$ with $v = 2$, $H330$ with $v = 3$, $H321$ with $v = \frac{5}{4}$ and finally, $H321$ *Predictive PID* with $v = 3$.

For the controllers with one negative β, we observe that the increase in the total number of windows is smaller in the combination test case than in the other test cases (refer to the bottom plots of Figure 4.9). This is because the analysis does not apply in the regions where the window-size drops or grows significantly over several windows due to a sudden increase or a sudden decrease in variation, respectively, in one or more of the coupled variables, and because the combination case has a greater number of such regions than the other test cases. The reason that the analysis does not apply in these regions is that successive window-sizes are not roughly equal, i.e., (4.3) does not hold.

4.6.4.2 Explaining Safety Factor Effect for LE

The top two bar graphs in Figure 4.10 show the raw change and percentage change in the number of rejected windows when γ is changed from $\gamma = 0.9$ to $\gamma = 0.7$ for all the controllers with LDIC and LE. The middle two and bottom two bar graphs in Figure 4.10 show the same for the *total* number of windows and the number of accepted windows, respectively. (The graphs for the number of accepted windows are included to explain something later in this section.)

Just as with CE, the number of rejected windows is smaller for $\gamma = 0.7$ than for $\gamma = 0.9$ in most cases. The reason for this is exactly the same as that given for CE in §4.6.4.1.

For the total number of windows, one can see that the values don't change significantly. For most, the change is less than 10%, while for a few, it is between 10% and around 20%. Next, we explain these observations. (Note: the steep drop cases for the $H321$ *PID Predictive* and $H321$ controllers should be disregarded, since the large difference in the total number of windows is due to the pathological case, described above in §4.6.3.3, occurring in the $\gamma = 0.9$ case or the $\gamma = 0.7$ case, but not both.)

For LE, as shown in A.3, if

$$H_{n-1} \approx H_n \tag{4.8}$$

then,

$$dev_n = c_n H_n^k \tag{4.9}$$

where H_n is the size of some window $[T_m, T_{m+1}]$, $k \in [1, 2]$ and $c_n = x''(\theta_t)$ for some $\theta_t \in [T_m, T_{m+1}]$.

If, for some constant c,

$$c_n \approx c \tag{4.10}$$

over several windows, then in that region,

$$dev_n \approx cH_n^k \tag{4.11}$$

Using (4.11) and the fact that the controller tries to make $dev_n \approx tol$,

$$H_n \approx H^* \tag{4.12}$$

Then, using (4.11), (4.12) and (3.8), the equation of the general third order controller for the LD method (considering just a single coupled variable) reduces to

$$\begin{aligned} H^* &= \left(\frac{\gamma \cdot tol}{cH^{*k}}\right)^{\beta_1} \left(\frac{tol}{cH^{*k}}\right)^{\beta_2} \left(\frac{tol}{cH^{*k}}\right)^{\beta_3} H^* \\ &= \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} \frac{H^*}{H^{*k(\beta_1+\beta_2+\beta_3)}} \\ &= \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} H^{*1-k(\beta_1+\beta_2+\beta_3)} \end{aligned} \tag{4.13}$$

$$H^{*k(\beta_1+\beta_2+\beta_3)} = \gamma^{\beta_1} \left(\frac{tol}{c}\right)^{\beta_1+\beta_2+\beta_3} \tag{4.14}$$

Therefore,

$$H^* = \gamma^{\frac{\beta_1}{k(\beta_1+\beta_2+\beta_3)}} \left(\frac{tol}{c}\right)^{\frac{1}{k}} \tag{4.15}$$

Of the notes that follow the derivation for CE in §4.6.4.1, notes 1 and 2 apply to LE as well. Note 3 does not apply as-is because, as we explained previously, for LE, if two successive windows contain an inflection point, it is *possible* for the endpoint deviation of the second window to be smaller than the deviations of one or more of its internal time-points, even if both windows are monotonic. However, as was also explained previously, there are no inflection points for three of the five test cases and there are only a few for each of the other two. As such, for all or the vast majority of the monotonic windows (which, remember, are the vast majority of all windows), the endpoint deviation is largest, and so, the same result, i.e., (4.15), will be obtained if internal points are included too.

From (4.15) we get,

$$\frac{H^*_{\gamma_2}}{H^*_{\gamma_1}} = \left(\frac{\gamma_2}{\gamma_1}\right)^{\frac{\beta_1}{2(\beta_1+\beta_2+\beta_3)}} \tag{4.16}$$

where $k = 2$ is used since that's the value we used for all tests with LE and

$H^*_{\gamma_1}$ is the window-size of the $n+1^{st}$ window when γ_1 is used

$H^*_{\gamma_2}$ is the window-size of the $n+1^{st}$ window when γ_2 is used

As stated previously, for about half of the controllers tested, all the β values are positive, and for the others, one and only one of β_2 and β_3 is negative (β_1 is positive for all controllers; see Table 3.1 and Table 3.2). For controllers of both these categories, $\frac{\beta_1}{2(\beta_1+\beta_2+\beta_3)} < 1$ in (4.16). That is necessarily true for the all-positive-βs case but not so for the one-negative-β case. However, for the particular β values for the controllers with one negative β value, $\frac{\beta_1}{2(\beta_1+\beta_2+\beta_3)}$ is still less than 1. As a result, with respect to (4.16), for all the controllers and for $\gamma_2 < \gamma_1$, when LE is used, the window-size is smaller when γ_2 is used than when γ_1 is used. In turn, that means that, for all the controllers, when LE is used, the number of accepted windows is larger when γ_2 is used than when γ_1 is used.

The results for the number of accepted windows as captured by the bottom two plots of Figure 4.10, however, do not match well with the model's predictions given above: the model predicts that the controllers H_0330, $H321$, $H321$ *Predictive PID* and H_0220 (to a lesser extent) will have a significantly greater number of accepted windows than the other controllers (3 to 15 times larger for the first three and 2 to 10 for H_0220) but this is not what we observe (somewhat true for two of the four controllers for the Sinusoids case only). The reason for this is that the model does not apply well for LE, unlike for CE. For example, for (4.15) to be true for the $n+1^{st}$ window, (4.8) needs to be true for the previous p_D windows but this is not the case for the majority of the windows for the LE tests (see the plots for $H110_0$ in Appendix B and the plots of the other controllers available via http://www.cs.toronto.edu/pub/reports/na/Rohan.MSc.Supp.2015.zip). The required model is more complicated.

4.6.5 Choice of Controller after Two Successively Accepted Windows following Rejected Window(s) for a $p_D = 3$ Controller

Suppose that $p_D = 3$ and that there are one or more consecutive rejections followed by an accepted window (to compute the next window's size in the rejections phase, one would use the window-size-selection-after-rejections policy detailed in §3.2.1). Then, as explained in §3.2.2, for $p_D = 2$ and $p_D = 3$ controllers, the next window's size should be computed using a $p_D = 1$ controller. Which $p_D = 1$ controller provides best performance? We did not carry out tests to determine this and simply chose the elementary controller, H_0110. One may carry out additional tests using other $p_D = 1$ controllers to determine which $p_D = 1$ controller/s, if any, provides the best performance.

As was also explained in §3.2.2, for $p_D = 3$ controllers, after two accepted windows following one or more consecutive rejected windows, one should use a $p_D = 1$ or $p_D = 2$ controller to compute the next window's size. Tests were run to determine if using H_0110 or a particular $p_D = 2$ controller provides better controller performance. Note that we did not consider $p_D = 1$ controllers other than H_0110 (one may want to carry out additional tests to determine which $p_D = 1$ controller/s, if any, provides the best performance.) For our test cases, no one controller provides significantly better performance for CE and $H211b$ with $b = 8$ provides the best performance for LE. This was determined by assessing the percentage change in the number of rejected windows for all test cases and all $p_D = 3$ controllers going from $H211b$ with $b = 8$ to each of the other $p_D = 2$ controllers and H_0110. Just the percentage changes aren't sufficient however. One also needs the raw change in the number of rejected windows because, for example, a 50% change could be due to 1 rejected window changing to 2 rejected windows. A difference

Figure 4.9: The top left and right plots are the percentage and raw differences, respectively, in the number of rejected windows when γ is decreased from $\gamma = 0.9$ to $\gamma = 0.7$ for all the test cases and all the controllers and for LDIC with CE. The bottom plots are the same except with the *total* number of windows instead of the number of rejected windows.

Figure 4.10: The top left and right plots are the percentage and raw differences, respectively, in the number of rejected windows when γ is decreased from $\gamma = 0.9$ to $\gamma = 0.7$ for all the test cases and all the controllers and for LDIC with LE. The middle and bottom plots are the same except with the *total* number of windows and number of accepted windows, respectively, instead of the number of rejected windows.

of a few in the number of rejected windows is not significant, as explained in §4.6.3.1, since that may be due to chance.

Clustered bar graphs of the percentage changes and the raw differences going from $H211b$ with $b = 8$ to H_0110 for LDIC with CE are shown in the top row of Figure 4.11. The same for LDIC with LE are shown in the bottom row of Figure 4.11. As can be seen in the top row of Figure 4.11, in the case of CE, there is no significant difference in the number of rejected windows going from $H211b$ with $b = 8$ to H_0110: the vast majority of cases have a raw difference less than or equal to 4 (the two exceptions are differences of only 5 and 6). In the case of LE, as one can see in the bottom row of Figure 4.11, there is a significant increase in the number of rejected windows going from $H211b$ with $b = 8$ to H_0110 for the sinusoids and combination test cases (for the combination test problem, an increase is seen only in some controllers), which means that $H211b$ with $b = 8$ provides better performance than H_0110.

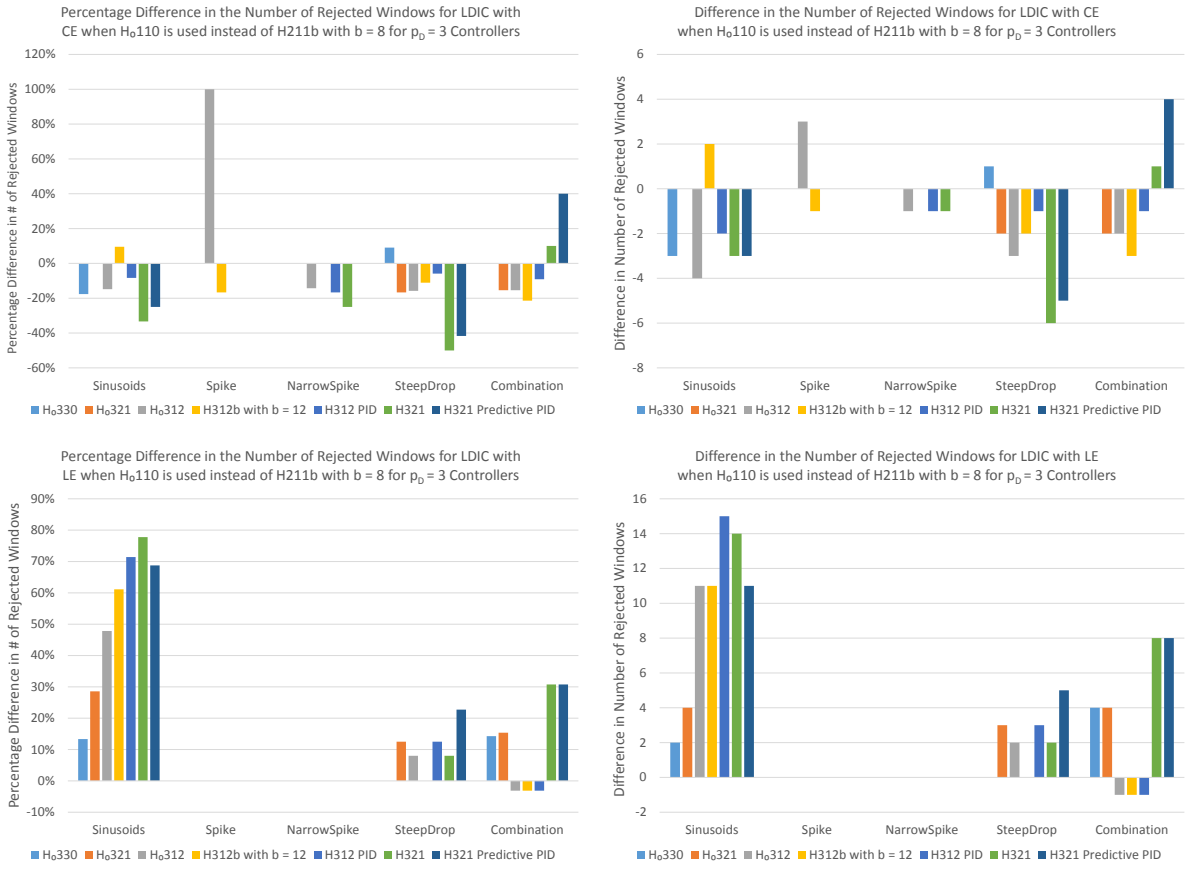

Figure 4.11: The top left and right plots are the percentage and raw differences, respectively, in the number of rejected windows when the controller is changed from $H211b$ with $b = 8$ to H_0110 for all the test cases and all the $p_D = 3$ controllers and for LDIC with CE. The bottom two are the same except for LE, instead of CE.

Chapter 5

Implementation Options

For the LD and LDIC methods, there are two main implementation options - *fully-distributed* and *fully-centralized* - with a range of possibilities in between. The distinction between these two implementation options is in the way in which the work is shared between the controller program and the codes, the work being the deviation estimation, window acceptability determination and window-size prediction. With the fully-distributed implementation, the work is shared whereas with the fully-centralized implementation, all of the work is done in the controller program. Note that it is possible to modify the fully-distributed implementation to make it less distributed and more centralized. Likewise, it is possible to modify the fully-centralized implementation to make it less centralized and more distributed. Note as well that the word controller will be used in this chapter to mean controller program and not window-size controller. Finally, note that, as stated previously in this document, we assume that, throughout the simulation, the values at the internal time-points of the most recent window are *not* saved by the codes.

5.1 Fully-Centralized LD and LDIC Implementation

The following three steps are repeated for each window until the end of the simulation is reached.

1. After executing a window, in the case of the LD method, each code sends the window endpoint values of all its coupled variables to the controller. In the case of the LDIC method, each code sends the values of all its coupled variables at *all* its time-points for the window. Then, all the codes wait.

2. Upon receiving the values of all the coupled variables of all the codes,

   (a) the controller computes the deviations of all the coupled variables using the values received from the codes and

      i. in the case of constant extrapolation, the values from the previous successful window's endpoint, which the controller needs to have stored throughout the simulation (for the first window it would use the initial values)

ii. in the case of linear extrapolation, the values from the endpoint of the previous two successful windows and the size of the previous window, which it needs to have stored throughout the simulation (for the first window, constant extrapolation will be used since there are no windows before it).

In the case of LD, the deviations are computed at just the window endpoint, whereas, in the case of LDIC, the deviations are computed at all the time-points. Then, the controller determines whether every deviation is less than or equal to the corresponding tolerance (the controller needs to have all the tolerances stored in its memory). If so, it sets the $acpt_{sys}$ Boolean to 1 (meaning the window is acceptable for the system of codes), else, it sets it to 0 (meaning the window is unacceptable for the system of codes).

(b) if $acpt_{sys} = 1$, the controller sends to each code the window endpoint values of the coupled variables that the code requires.

(c) the controller determines H_{n+1}, the next window's size, using formulas (3.8), (3.9) and (3.10) in the case of LD or (3.11), (3.9) and (3.10) in the case of LDIC.

(d) the controller sends $acpt_{sys}$ and H_{n+1} to all of the codes.

3. Upon receiving $acpt_{sys}$ and H_{n+1} from the controller, each code resumes the simulation in one of two ways. If the window was deemed unacceptable ($acpt_{sys} = 0$), it returns to the window's starting point and executes a window of size H_{n+1} (using the same values for the coupled variables at the window starting point that it used for the rejected window). On the other hand, if the window was deemed acceptable ($acpt_{sys} = 1$), it resumes execution from its current position over a window of size H_{n+1}. (In doing so, it will use the window endpoint values sent in step 2b. We are assuming that these window endpoint values arrive at all the codes before $acpt_{sys}$ does since the window endpoint values are sent before $acpt_{sys}$ (steps 2b and 2d respectively). If this cannot be gauranteed, each code should check that it has received the window endpoint values of all the coupled variables it needs before proceeding to execute the next window.)

5.2 Fully Distributed LD and LDIC Implementations

The following four steps are repeated until the end of the simulation is reached.

1. After executing each window, each code performs the following three steps.

(a) Each code c computes the deviations of all its coupled variables at the window endpoint (LD) or at all the time-points (LDIC) using

i. in the case of constant extrapolation, the values from the previous successful window's endpoint, which c needs to have stored throughout the simulation (for the first window it would use the initial values)

ii. in the case of linear extrapolation, the values from the endpoint of the previous two successful windows and the size of the previous window, which c needs to have stored

throughout the simulation (for the first window, constant extrapolation will be used since there are no windows before it).

(b) Each code applies the controller formula to all its coupled variables (for LD, at the window endpoint only and for LDIC, at all its time-points) and then, determines the minimum among the resulting numbers. It then imposes the relative and absolute limits to compute the size of the next window that is best for it. That is, each code c, for $c = 1, \ldots, C$, where C is the number of codes in the system, computes H^c_{n+1} as follows.

For LD,

$$H'_c = \min \left\{ \left(\frac{\gamma \cdot tol^{i,c}}{dev_n^{i,l^c_n,c}}\right)^{\beta_1} \left(\frac{tol^{i,c}}{dev_{n-1}^{i,l^c_{n-1},c}}\right)^{\beta_2} \left(\frac{tol^{i,c}}{dev_{n-2}^{i,l^c_{n-2},c}}\right)^{\beta_3} \left(\frac{H_n}{H_{n-1}}\right)^{-\alpha_2} \left(\frac{H_{n-1}}{H_{n-2}}\right)^{-\alpha_3} H_n \ : \ i = 1, ..., r^c \right\} \tag{5.1}$$

$$H''_c = \min\left\{\max\left\{H'_c, 0.5H_n\right\}, 2H_n\right\} \tag{5.2}$$

$$H^c_{n+1} = \min\left(\max\left(H''_c, H_{min}\right), H_{max}\right) \tag{5.3}$$

where all terms, except the new ones H'_c, H''_c and H^c_{n+1}, are the same as the corresponding ones in (3.8), (3.9) and (3.10); the new terms are explained below

H'_c is the minimum among the next window-size values computed for all the coupled variables of code c only

H''_c is the result of applying the relative limits on H'_c

H^c_{n+1} is the final value for the next window-size *for* code c, obtained by imposing a second set of limits, which are absolute.

For LDIC,

$$H'_c = \min \left\{ \left(\frac{\gamma \cdot tol^{i,c}}{\max\limits_{j=1...l^c_n}\{dev_n^{i,j,c}\}}\right)^{\beta_1} \left(\frac{tol^{i,c}}{\max\limits_{j=1...l^c_{n-1}}\{dev_{n-1}^{i,j,c}\}}\right)^{\beta_2} \times \left(\frac{tol^{i,c}}{\max\limits_{j=1...l^c_{n-2}}\{dev_{n-2}^{i,j,c}\}}\right)^{\beta_3} \left(\frac{H_n}{H_{n-1}}\right)^{-\alpha_2} \left(\frac{H_{n-1}}{H_{n-2}}\right)^{-\alpha_3} H_n : i = 1, ..., r^c \right\} \tag{5.4}$$

and (5.2) and (5.3) are used to compute H^c_{n+1}, where all terms are the same as those in (3.11), (3.9) and (3.10), except for the new terms H'_c, H''_c and H^c_{n+1}, which are explained above (under the LD equations)

(c) Each code c checks if the deviations of all its coupled variables at the window endpoint (LD) or at all the time-points (LDIC) are less than or equal to the corresponding tolerances. If so, c sets the Boolean $acpt_c$ to 1 (meaning the window is acceptable for code c), else, it sets it to 0 (meaning the window is unacceptable for code c)

(d) Each code c sends $acpt_c$ and H^c_{n+1} to the controller.

2. (a) Upon receiving $acpt_c$ from all the codes, the controller checks if each and every $acpt_c$ is equal to 1. If they are, it sets the Boolean $acpt_{sys}$ to 1 (meaning the window is acceptable for all the codes in the system), else, it sets it to 0 (meaning the window is unacceptable for one or more codes in the system).

   (b) Upon receiving H^c_{n+1} from all the codes, the controller determines the next window's size by taking the minimum among all the H^c_{n+1} values, i.e., it computes the following

$$H_{n+1} = \min_c \{\ H^c_{n+1}\ :\ c = 1, ..., C\ \} \tag{5.5}$$

   (c) The controller sends $acpt_{sys}$ and H_{n+1} to all the codes.

3. Upon receiving $acpt_{sys}$ from the controller, if $acpt_{sys} = 1$, each code c_1 sends to every other code c_2 the window endpoint values of those coupled variables computed by c_1 that c_2 requires, if any. (If $acpt_{sys} = 0$, the codes do not need to send the values of their coupled variables to the other codes.)

4. Upon receiving $acpt_{sys}$ and H_{n+1} from the controller, each code resumes the simulation in one of two ways. If the window was deemed unacceptable ($acpt_{sys} = 0$), it returns to the window's starting point and executes a window of size H_{n+1} (using the same values for the coupled variables at the window starting point that it used for the rejected window). On the other hand, if the window was deemed acceptable ($acpt_{sys} = 1$), it first checks if it has received the window endpoint values for all the coupled variables it requires (which were sent in step 3). If it has not, it waits. Once it has received all these values, it resumes execution from its current position over a window of size H_{n+1}.

5.3 Centralized versus Distributed Implementations

As mentioned above, it is possible to make the fully centralized implementation less centralized and more distributed. This can be done to varying degrees. For example, assuming the LD method is used, instead of the codes sending the window endpoint values to the controller and the controller computing the deviations, the codes could compute the deviations themselves and send the deviations to the controller. One could make the implementation even more distributed by having each code check if all its deviations are less than the corresponding tolerances and send the Boolean result to the controller (which will then have to determine if all the codes sent a Boolean 1) instead of the controller checking the deviations of all the coupled variables of all the codes. In this way, i.e., incrementally, the centralized implementation can be modified to make it more and more distributed. Similarly, the fully distributed implementation can be made less distributed and more centralized, to varying degrees.

The more centralized the implementation, the lesser the degree of code modification required and the greater the ease of maintenance. On the other hand, the more centralized the implementation, the greater the data transfer time and the time to compute the deviations, $acpt_{sys}$ and H_{n+1}. These

differences in the data transfer and computing time will not be significant for systems that have one or more complex codes, such as those of AECL. These points are explained below.

First, consider the degree of code modification required. The centralized and distributed implementations both require modifications to enforce pausing the execution after completion of a window, responding appropriately depending on the value of $acpt_{sys}$ (step 3 in §5.1 and steps 3 and 4 in §5.2), etc. However, the distributed implementation requires some extra additions/modifications. First, it requires the codes to compute H^c_{n+1} and $acpt_c$ (steps 1b and 1c, respectively, in §5.2). Second, it requires the codes to know to which codes to send which of its coupled variables and when to send them (step 3 in §5.2). So, the centralized implementation requires significantly less code modification. For nuclear simulation codes, minimal or no change to the codes is preferred because of the significant verification and validation (V&V) work that must be performed after any changes made. That will be expensive both monetarily (equivalently, man-hours) and in the length of time required.

Second, consider ease of maintenance. With the distributed implementation, if any changes are made to the coupling method employed (LD or LDIC), it would have to be duplicated in all the codes. With the centralized implementation on the other hand, the changes need to be made only in the controller. This is a significant advantage of the centralized implementation.

Third, consider the length of time required for data transfer, which affects the simulation completion time (the greater the data transfer time, the greater the simulation completion time). In the centralized implementation case, for the LD method, all the codes need to send the window endpoint values of their coupled variables to the controller for all the windows, i.e., regardless of whether the window was rejected or accepted; likewise, for the LDIC method, all the codes need to send the values of their coupled variables for all the time-points to the controller for all the windows. With the distributed implementation on the other hand, for both the LD and LDIC methods, the codes need to send only the window endpoint values and only if the window was accepted! So, the amount of data transferred in the centralized implementation case is larger and so, the time taken for the transfer is greater. In addition, the centralized implementation has the disadvantage that all the data transfers are to and from a single point, i.e., the controller, instead of many to many, as is the case for the distributed approach. This makes the data transfer take longer for the centralized approach than the distributed approach. Note that this increase in simulation completion time due to the longer data transfer time may be insignificant: the time taken by one or more codes to execute a window may be far greater than the data transfer time saved by using the distributed implementation; this is true for systems that contain complex codes, such as those of AECL.

Fourth and finally, consider the time required to compute the deviations, $acpt_{sys}$ and H_{n+1}. In the case of the centralized implementation, all the work is done by the controller whereas in the distributed case, the work is done in parallel by the codes and then, the global result is determined by the controller. So, the computing time for the quantities stated above is shorter in the case of the distributed implementation. Note that this difference may not be significant: the time taken by one or more codes to execute a window may be far greater than the time saved in computing the quantities stated above using the distributed implementation; this is true for systems that contain complex codes, such as those of AECL.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Regardless of the controller, when LE is used, as opposed to CE, the average window-size is significantly larger. On the other hand, when LE is used, the number of rejected windows is significantly larger. Overall, the total number of windows is significantly smaller when LE is used.

The LD method is unreliable, unlike the LDIC method. For example, it may miss spikes of size greater than the deviation tolerance within a window. Otherwise, the LD and LDIC methods produce similar results for our simple test cases (all variables are mostly monotonic and have few inflection points).

When the operator splitting method is used to simulate a multi-component system, the difference in the overall computational work required or in the simulation computing time, between two controllers, is determined primarily by the the difference in their number of rejected windows, provided that the three assumptions stated in §4.6.3 hold (and assuming that each code does not store its coupled variables' values from the most recent window in memory).

When CE is used with LDIC, $H321$ does better than the other controllers in terms of minimizing the number of rejected windows over the five test cases. When LE is used with LDIC, no one controller is best, but three controllers (H_0110, H_0211 and $H312$ PID) perform significantly worse than the others for the sinusoids test case.

Some controllers fail to capture slowly varying coupled variables without rejected windows (e.g., $H211PI$). Some $p_D = 3$ controllers exhibit pathological oscillatory behaviour in the window-size and others exhibit a significant dip in the window-size followed by a monotonic increase in regions where a monotonic increase throughout is expected and preferred.

We consider these results preliminary in that they do not fully take into account the effects of coupling between the codes. Therefore, we strongly recommend that further testing of the LD and LDIC methods be performed on test problems that exhibit the coupling expected of the systems that the reader wishes to simulate.

Implementation options were described in detail. There are two main options, fully distributed and fully centralized, and a range of possibilities in between. The right choice depends on the characteristics of the particular multi-component system and the emphasis placed on ease of code maintenance, etc.

6.2 Future Work

As mentioned previously, we strongly recommend that further testing be done on problems that *include coupling* between the variables. Also, we recommend that further testing be done with problems that are similar to the system the reader wishes to simulate. For nuclear reactor simulations, to accomplish both these suggestions simultaneously, simplified reactor models, such as those in [7], can be used. Simplified reactor models are of course simpler than real reactor systems. As such, while the simplified models option is significantly better than using arbitrary elementary test problems, like we did in this work, it does not capture all the characteristics and difficulties of the real reactor simulation problem.

One may want to carry out tests to determine the effect of the following on the results:

1. extrapolation methods of higher-order than CE and LE,
2. controllers not considered in this work.

Appendix A

Proofs

A.1 Derivation of the Elementary Controller Formula

For a k-th order numerical method for ODEs, it is standard to assume the error per unit step estimate for the n^{th} step satisfies

$$est_n = c_n h_n^k \tag{A.1}$$

Now suppose that we've completed the n^{th} step and we know both est_n and h_n. Thus, we also know

$$c_n = \frac{est_n}{h_n^k} \tag{A.2}$$

Before taking the next step, we don't know est_{n+1}, h_{n+1} or c_{n+1}. However, these three variables should be related by

$$est_{n+1} = c_{n+1} h_{n+1}^k \tag{A.3}$$

Our goal is to choose h_{n+1} so that $est_{n+1} \approx tol$. The term c_n depends on the problem and should change slowly if we are taking small steps. So, it is reasonable to assume

$$c_{n+1} \approx c_n \tag{A.4}$$

Using (A.2) and (A.4), we get

$$c_{n+1} \approx \frac{est_n}{h_n^k} \tag{A.5}$$

Substituting (A.5) into (A.3), we get

$$est_{n+1} \approx \frac{est_n}{h_n^k} h_{n+1}^k \tag{A.6}$$

Since as noted above, we want $est_{n+1} \approx tol$, (A.6) suggests choosing h_{n+1} so that

$$tol \approx \frac{est_n}{h_n^k} h_{n+1}^k \tag{A.7}$$

Re-arranging (A.7), we get

$$h_{n+1} \approx \left(\frac{tol}{est_n}\right)^{\frac{1}{k}} h_n \tag{A.8}$$

Note that we've been using $\approx$ throughout the derivation. So, it is not reasonable to expect (A.8) to hold exactly. Therefore, it is common to introduce a safety factor $\gamma < 1$ to avoid too many failed steps and use the formula

$$h_{n+1} = \left(\frac{\gamma \cdot tol}{est_n}\right)^{\frac{1}{k}} h_n \tag{A.9}$$

instead of (A.8). This is the classical step-size selection algorithm for ODE solvers, known as the elementary controller.

A.2 Choice of k for Constant Extrapolation

Assume that a coupled variable $x(t)$ is smooth enough that $x'(t)$ exists and is continuous for $t \in [T_i, T_{i+1}]$. Then,

$$x(t) = x(T_i) + x'(\theta_t)(t - T_i) \tag{A.10}$$

for some $\theta_t \in [T_i, t]$. θ_t changes as t changes in (A.10), hence the subscript t in θ_t.

Now, for constant extrapolation,

$$\tilde{x}(t) = x(T_i) \tag{A.11}$$

for all $t \in [T_i, T_{i+1}]$. Therefore, the deviation satisfies

$$x(t) - \tilde{x}(t) = x'(\theta_t)(t - T_i) \tag{A.12}$$

Hence, if $x'(\theta_t)$ does not vary too much for $t \in [T_i, T_{i+1}]$, we see that the deviation grows approximately linearly with t. Therefore, in particular, if we let $H_i = T_{i+1} - T_i$, we see that

$$dev_i \approx c_i H_i \tag{A.13}$$

where $c_i = x'(\theta_t)$ for some $\theta_t \in [T_i, T_{i+1}]$. Equation (A.13) is similar to (A.1) with $k = 1$. Therefore, for constant extrapolation, we use $k = 1$ in the step-size controller formulae.

A.3 Choice of k for Linear Extrapolation

Assume that a coupled variable $x(t)$ is smooth enough that $x''(t)$ exists and is continuous for $t \in [T_{i-1}, T_{i+1}]$. Then,

$$x(t) = x(T_i) + x'(T_i)(t - T_i) + \frac{x''(\theta_t)}{2}(t - T_i)^2 \tag{A.14}$$

for some $\theta_t \in [T_i, t]$. θ_t changes as t changes in (A.14), hence the subscript t in θ_t.

For linear extrapolation,

$$\tilde{x}(t) = x(T_i) + s(t - T_i) \tag{A.15}$$

$$s = \frac{x(T_i) - x(T_{i-1})}{T_i - T_{i-1}} \tag{A.16}$$

for $t \in [T_i, T_{i+1}]$. Therefore, the deviation satisfies

$$x(t) - \tilde{x}(t) = (x'(T_i) - s)(t - T_i) + \frac{x''(\theta_t)}{2}(t - T_i)^2 \tag{A.17}$$

Now, note that

$$s = \frac{x(T_i) - x(T_{i-1})}{T_i - T_{i-1}} = x'(\hat{T}) \tag{A.18}$$

for some $\hat{T} \in [T_{i-1}, T_i]$ and

$$x'(T_i) - x'(\hat{T}) = x''(\tilde{T})(T_i - \hat{T}) \tag{A.19}$$

for some $\tilde{T} \in [\hat{T}, T_i] \subset [T_{i-1}, T_i]$. Thus, from (A.17), (A.18) and (A.19), we see that

$$x(t) - \tilde{x}(t) = x''(\tilde{T})(T_i - \hat{T})(t - T_i) + \frac{x''(\theta_t)}{2}(t - T_i)^2 \tag{A.20}$$

If we let $H_{i-1} = T_i - T_{i-1}$, $\rho H_{i-1} = T_i - \hat{T}$ and $H_i = T_{i+1} - T_i$, we see from (A.20) that the deviation over the next step from T_i to T_{i+1} satisfies

$$dev_i \approx c_i^{(1)} \rho H_{i-1} H_i + c_i^{(2)} H_i^2 \tag{A.21}$$

Hence, if $H_{i-1} \approx H_i$ (as is usually the case) and $\rho c_i^{(1)} \ll c_i^{(2)}$, then the deviation will vary approximately quadratically with H_i. On the other hand, if $H_{i-1} \approx H_i$ and $c_i^{(2)} \ll \rho c_i^{(1)}$, then the deviation will vary approximately linearly with H_i. For most steps, we should have $\rho c_i^{(1)} \approx c_i^{(2)}$, in which case the deviation will vary as H_i^k for some k between 1 and 2. So, when using linear extrapolation, one should use $k \in [1, 2]$.

Appendix B

Plots of H_0110, the Elementary Controller

The 2×2 plots (see 4.3 for an explanation of the term) of all the controllers cannot be included in this document as there are too many plots (22 unique controllers were tested and there are 10 plots for each controller; thus, there are 220 2×2 plots). However, some plots are included in Chapter 4 to explain various points. The plots of *all* the controllers tested can be found in the folder *cdt/results/*, where the folder *cdt/* is zipped up inside *http://www.cs.toronto.edu/pub/reports/na/Rohan.MSc.Supp.2015.zip*. For a description of how the results are structured in the *cdt/results/* folder, see §4.5.2.

Here we provide the plots for H_0110, the Elementary Controller (corresponding to (3.1)), as an illustration.

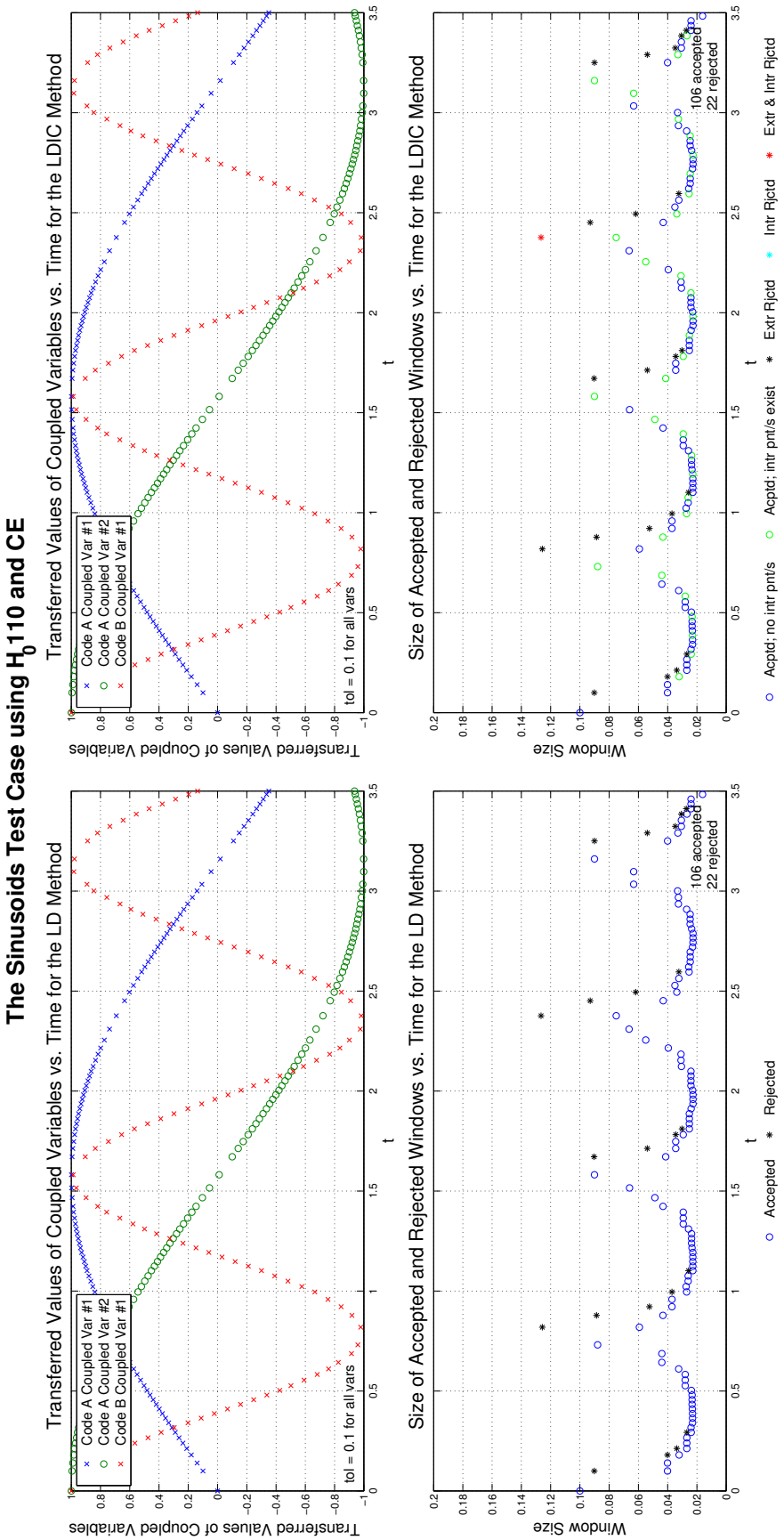

Figure B.1: H_0110 Sinusoids

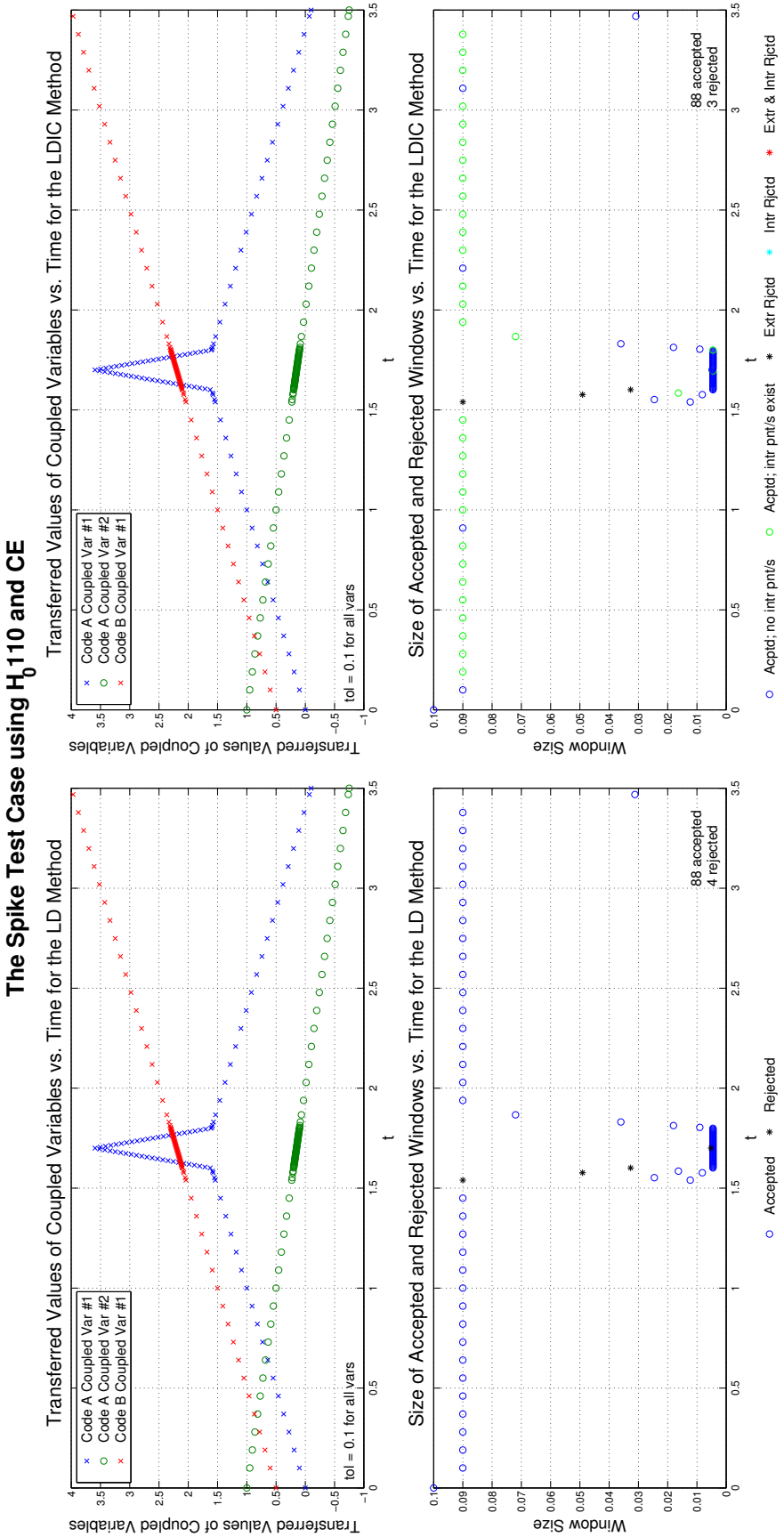

Figure B.2: H_0110 Spike

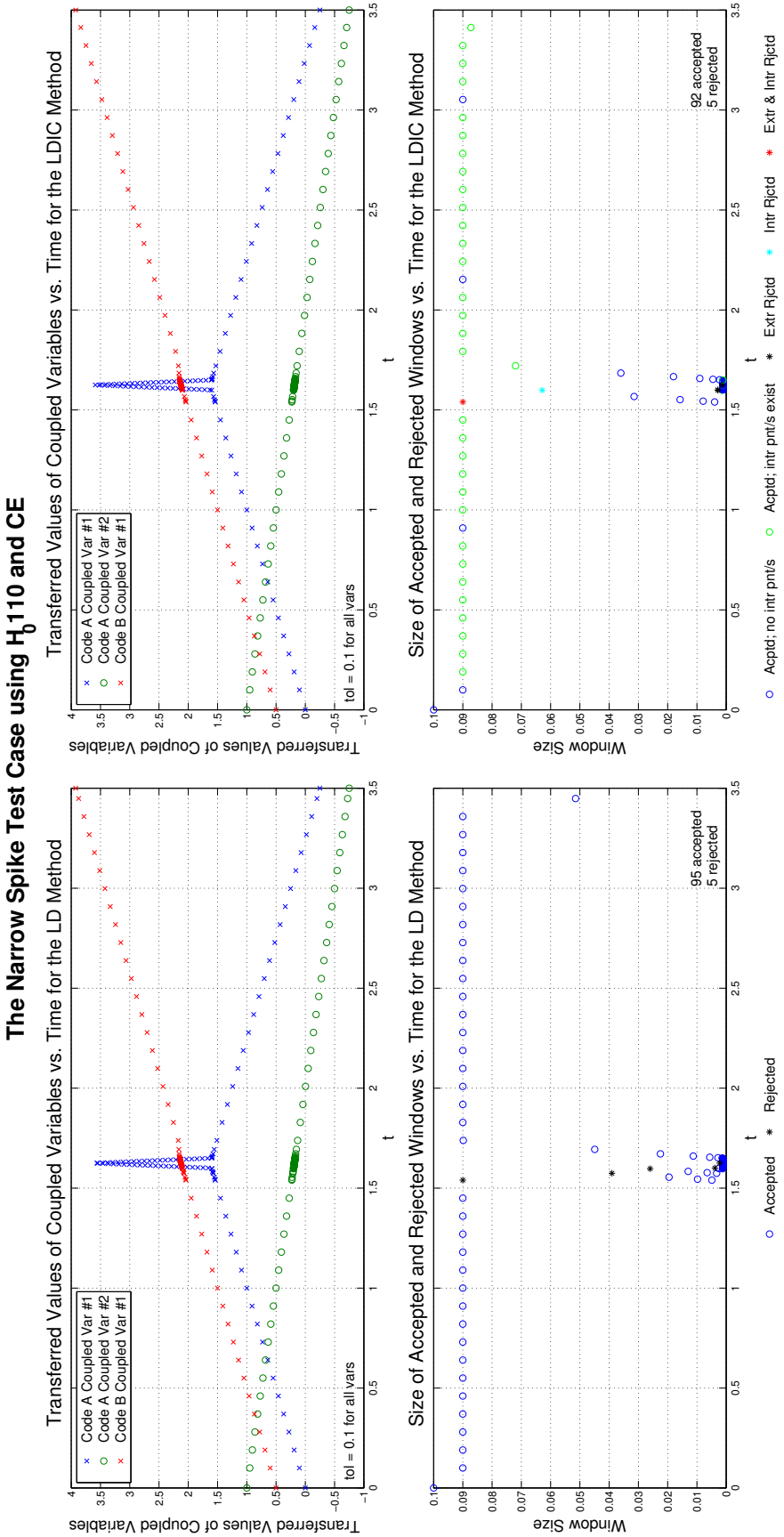

Figure B.3: H_0110 Narrow Spike

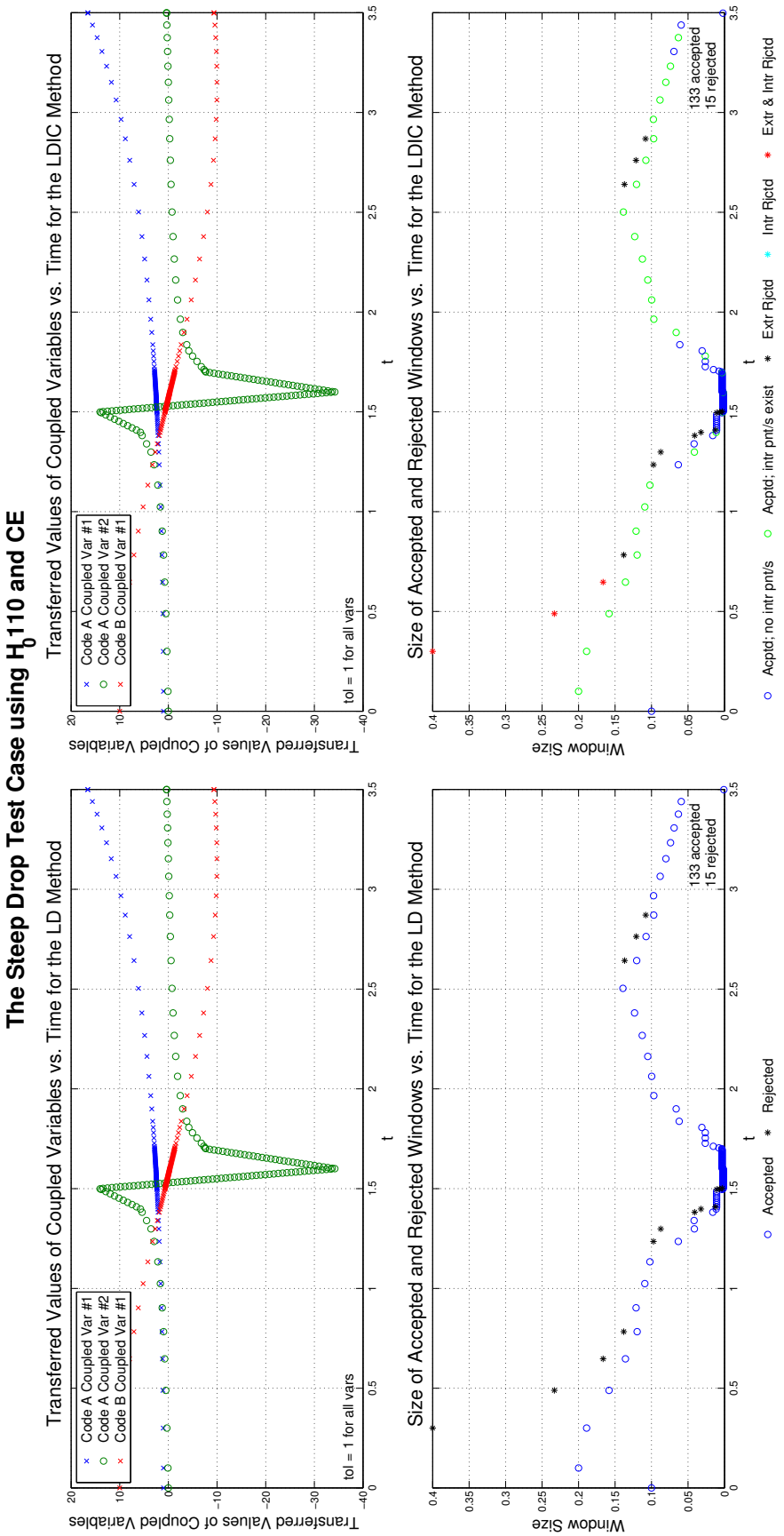

Figure B.4: H_0110 Steep Drop

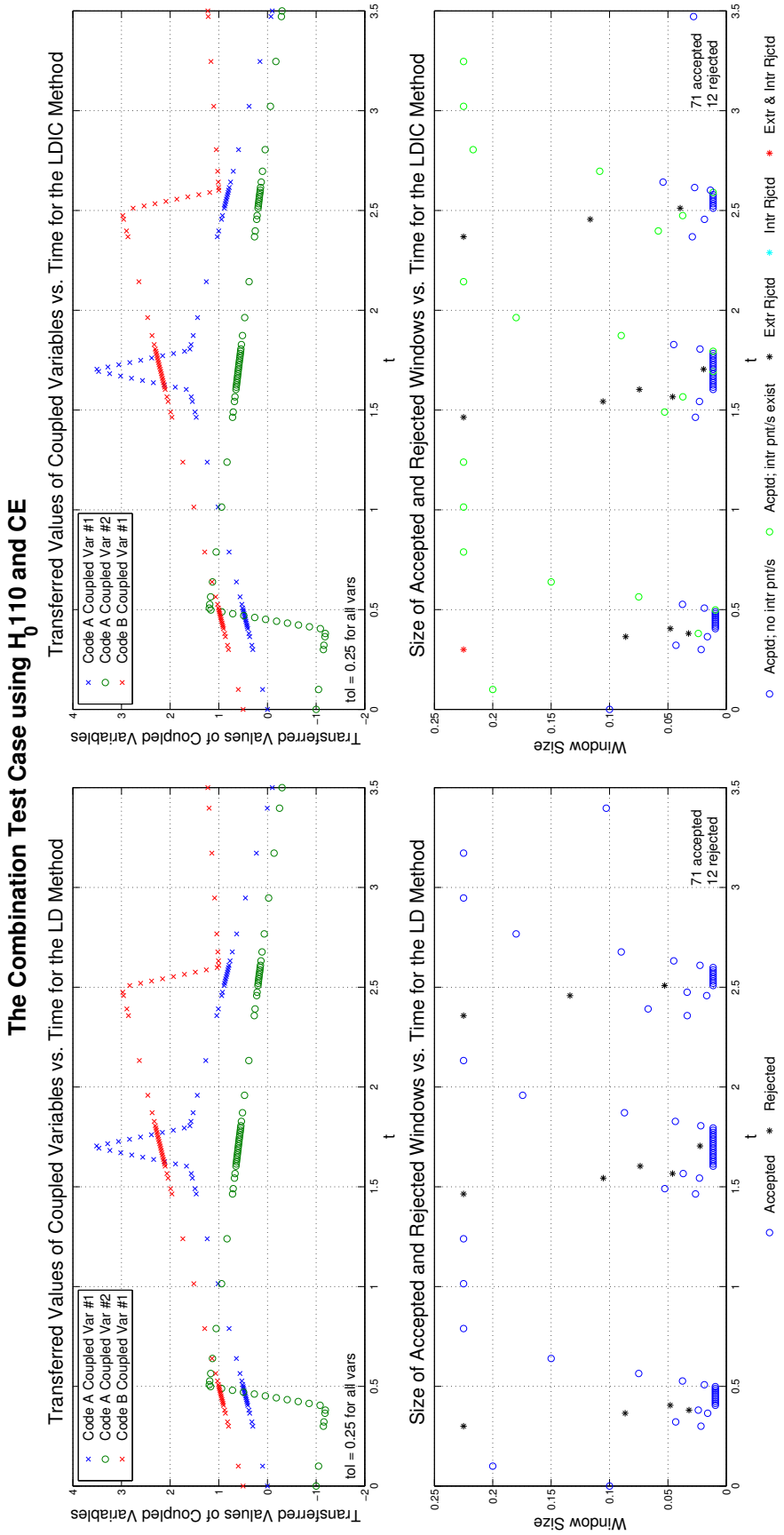

Figure B.5: H_0110 Combination

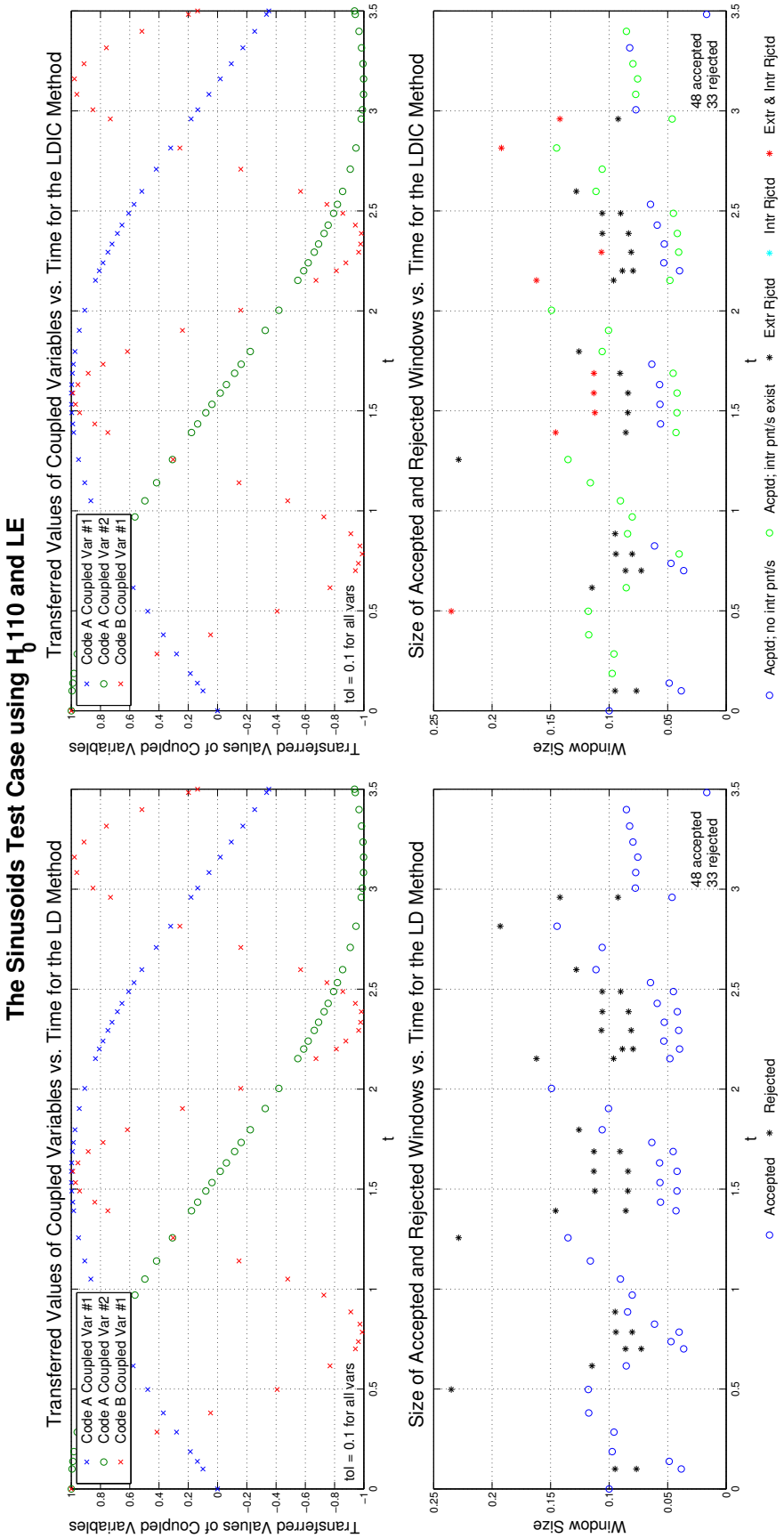

Figure B.6: H_0110 Sinusoids

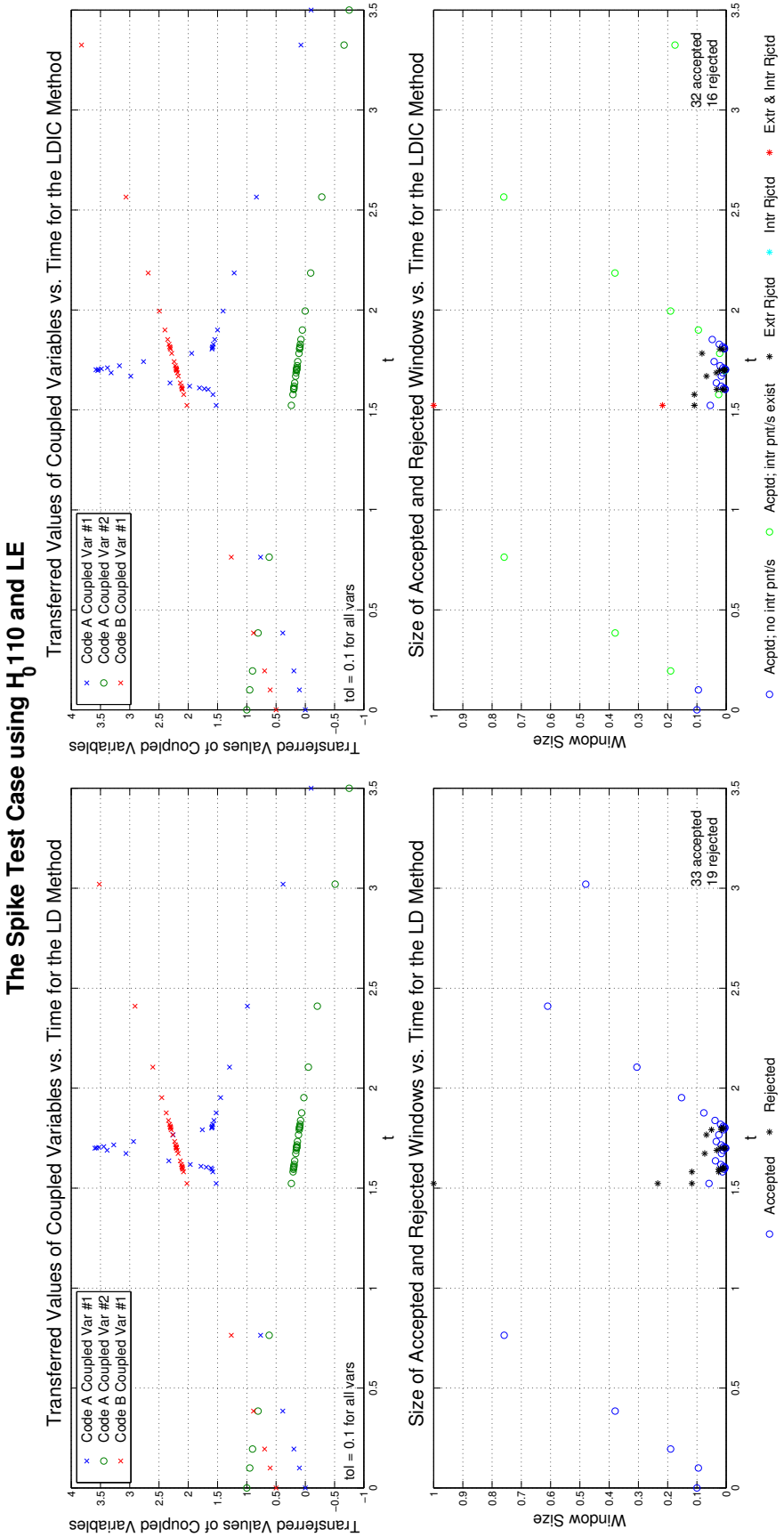

Figure B.7: H_0110 Spike

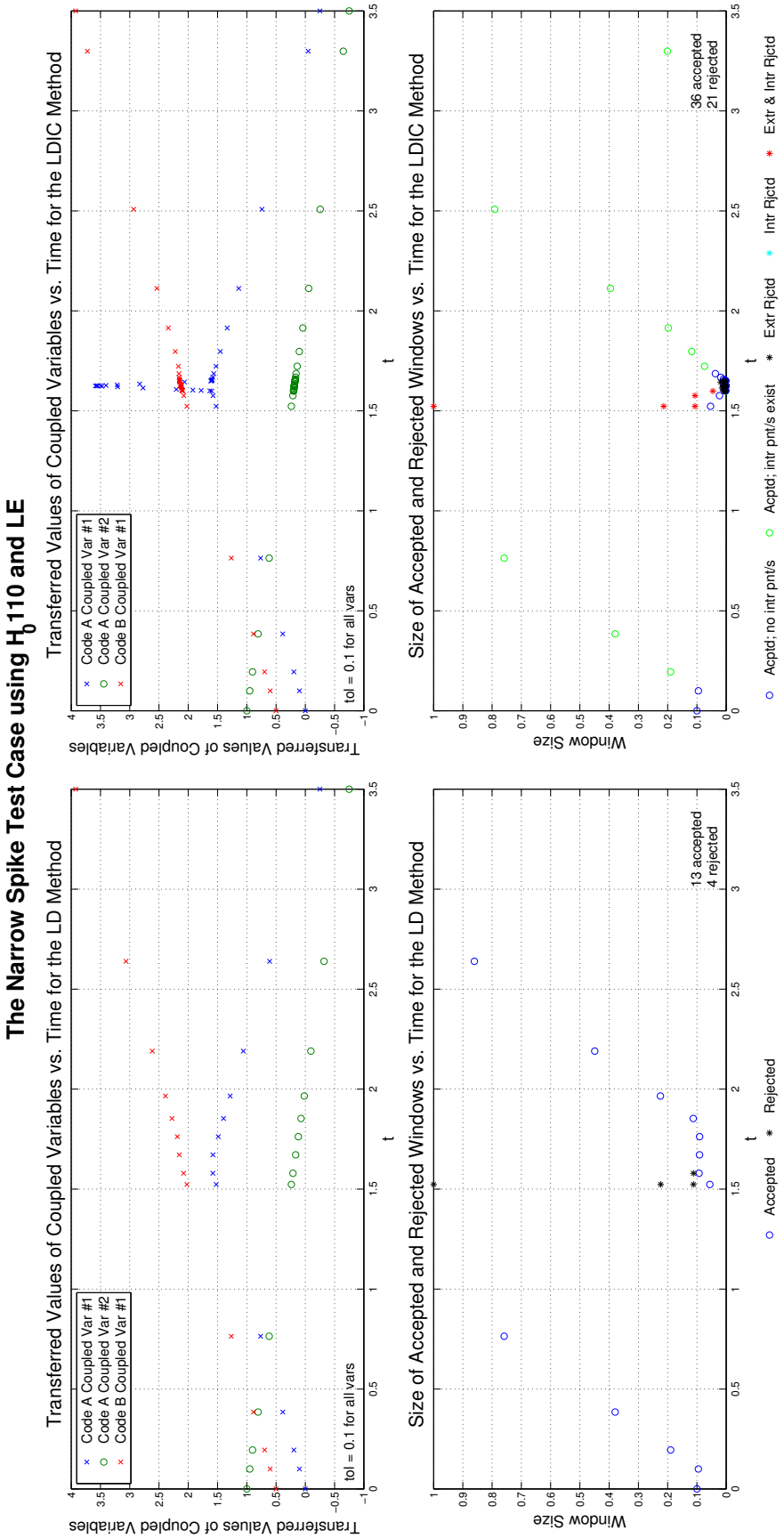

Figure B.8: H_0110 Narrow Spike

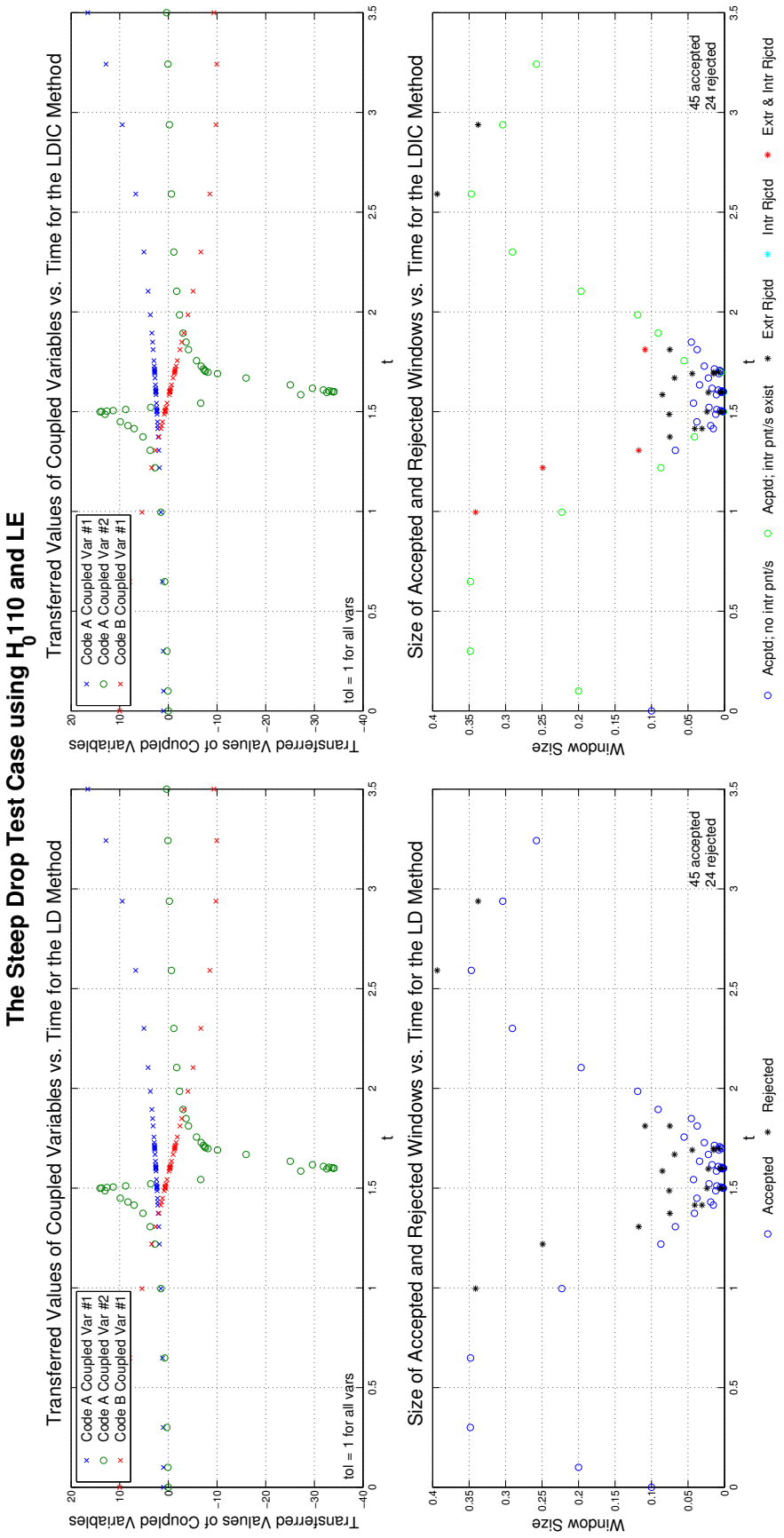

Figure B.9: H_0110 Steep Drop

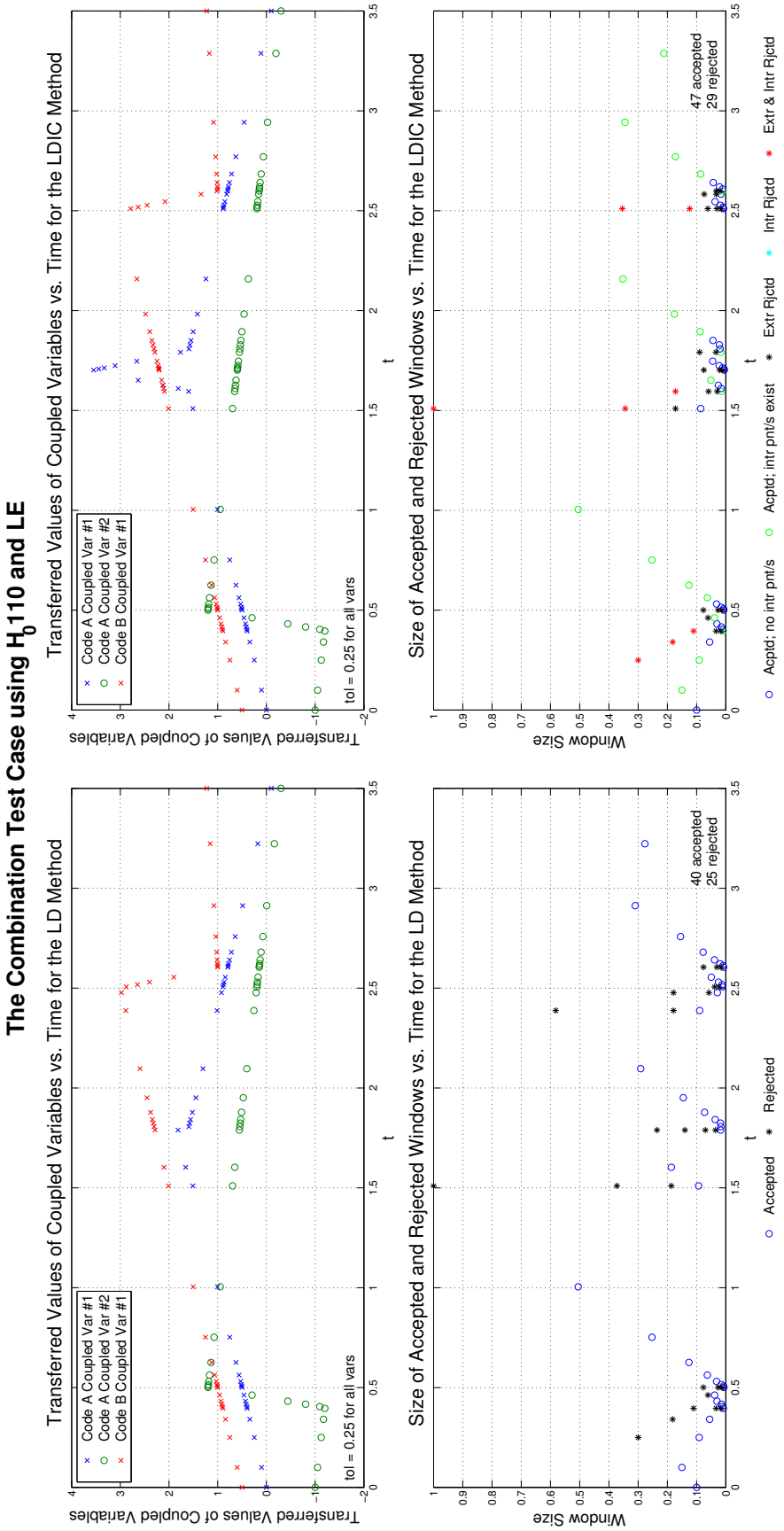

Figure B.10: H_0110 Combination

Bibliography

[1] ADINA R&D, Inc. Adina multiphysics. `http://www.adina.com/multiphysics.shtml`, 2013. Visited on 2014-05-22.

[2] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[3] Kevin Burrage. *Parallel and Sequential Methods for Ordinary Differential Equations*. Oxford Science Publications, 1995.

[4] David F. Griffiths and Desmond J. Higham. *Numerical Methods for Ordinary Differential Equations. Initial Value Problems*. Springer-Verlag, 2010.

[5] Kjell Gustafsson, Michael Lundh, and Gustaf Soderlind. A PI stepsize control for the numerical solution of ordinary differential equations. *BIT*, 28(2):270–287, 1988.

[6] C. Housiadas. Lumped parameters analysis of coupled kinetics and thermal-hydraulics for small reactors. *Annals of Nuclear Energy*, pages 1315–1325, 2002.

[7] Jean C. Ragusa and Vijay S. Mahadevan. Consistent and accurate schemes for coupled neutronics thermal-hydraulics reactor analysis. *Nuclear Engineering and Design*, pages 566–579, 2009.

[8] Gustaf Soderlind. Automatic control and adaptive time-stepping. *Numerical Algorithms*, pages 281–310, 2002.

[9] Gustaf Soderlind. Digital filters in adaptive time-stepping. *ACM Transactions on Mathematical Software*, 29(1):1–26, 2003.

[10] Omar Zerkak, Ivan Gajev, Annalisa Manera, Tomasz Kozlowski, Andre Gommlich, Stefanie Zimmer, Sören Kliem, Nicolas Crouzet, and Martin Zimmermann. Revisiting temporal accuracy in neutronics/TH code coupling using the NURESIM LWR simulation platform. In *The 14th International Topical Meeting on Nuclear Reactor Thermalhydraulics, NURETH-14*, Toronto, Canada, September 2011.