Fast Time Domain Simulation for Large Order Hybrid Systems

by

Kin Cheong Sou

B.S., Nanjing University of Aeronautics and Astronautics, Nanjing, China, 2000

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Author .....
Department of Aeronautics and Astronautics
May 23, 2002

Certified by .....
Professor Olivier L. de Weck
Assistant Professor
Thesis Supervisor

Read by .....
Professor Wallace E. Vander Velde
Professor Emeritus
Thesis Reader

Accepted by .....
Professor Wallace E. Vander Velde
Chairman, Department Committee on Graduate Students

Fast Time Domain Simulation for Large Order Hybrid Systems

by

Kin Cheong Sou

Submitted to the Department of Aeronautics and Astronautics
on May 23, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Simulation is an important tool for the analysis and design of complex systems. As the models become more and more complex, more powerful simulation methods are desired. As an attempt to address this problem, a simulation scheme is proposed and developed in this thesis. The main objective of this work is to simulate continuous-time linear time-invariant (CT-LTI) systems efficiently with acceptable approximation and error.

The method basically divides the original large order system into smaller subsystems, simulates them with state transition formula and superposes the responses by using the linearity property of LTI systems. A similarity transformation can first be employed to obtain the modal canonical form of the system. The modal form can then be divided into subsystems with manageable sizes. Discretization scheme specially developed for these subsystems can then be employed to get their discretized counterparts. At this stage, the original continuous-time IVP becomes a set of of smaller matrix vector multiplication routines. Special matrix vector product solver is chosen to exploit the sparsity resulted from the diagonalized structure of the A-matrix. Also, subsystems are considered in frequency domain to see if multiple sampling rate scheme is acceptable. Finally, the results of the simulations can be superposed to form the response of the original system. Next, the more advanced problem of simulating hybrid feedback control systems are studied.

The algorithm has been compared with the standard MATLAB LTI system simulation routine `lsim.m` and is shown to be able to simulate large order systems even when `lsim.m` fails. It is also shown that the new simulator is more efficient than `lsim.m` even for moderately large order systems simulations. Finally, the method is applied to SIM (Space Interferometry Mission, 2000 state variables) as an example of real world applications.

Thesis Supervisor: Professor Olivier L. de Weck
Title: Assistant Professor

Acknowledgements

I would like to thank my advisor Prof. Olivier L. de Weck for his support, guidance and especially trust. Furthermore, he gave me the opportunity to conduct research independently. My parents cannot be thanked enough for their support, concern and expectations. Their encouragement smoothed my harsh transition from undergrad study in China to graduate study in the United States and stimulated my desire for future study. Chinese friend Yong Shi's help during my hard time at M.I.T. is also appreciated. Alice Liu is thanked for introducing me to Oli, my advisor. My office mate, Kyungyeol Song is acknowledged for sharing his experience in control with me. Prof. Wallace E. Vander Velde's meticulous reading of my thesis and his practical advises are highly appreciated. Finally, Cyrus Jilla is thanked, simply for being a nice guy at the Space System Laboratory.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

AA	anti-aliasing
ACS	attitude control system
CPU	central processing unit
CT	continuous-time
DC	direct current
DSF	downsampling factor
DT	discrete-time
DTFT	discrete-time Fourier transform
DTSS	discrete-time state space
FEM	finite element model
FFT	fast Fourier transform
FIR	finite impulse response
FLOP	floating point operation
FOH	first order hold
FT	Fourier transform
HPF	highpass filter
HST	Hubble Space Telescope
IFT	inverse Fourier transform
IIR	infinite impulse response
IVP	initial value problem
JPL	Jet Propulsion Laboratory
LPF	lowpass filter
LQG	linear quadratic Gaussian (dynamic compensator)
LQR	linear quadratic regulator

LTI	linear time-invariant
MATLAB	Matrix Laboratory
MIMO	multi-input-multi-output
NASA	National Aeronautics and Space Administration
NGST	Next Generation Space Telescope
NMZ	nonminimum-phase zero
ODE	ordinary differential equation
OPD	optical path difference
PDE	partial differential equation
PID	proportional-integral-derivative (controller)
PSD	power spectral density
PSS	precision support structure
RAM	random access memory
RK4	Runge Kutta 4^{th} order
RMS	root-mean-square
RSS	root-sum-square
RV	random variable
RWA	reaction wheel assembly
SIM	Space Interferometry Mission
SISO	single-input-single-output
TPF	Terrestrial Planet Finder
VLSI	very large scale integration
WFT	wavefront tilt
ZOH	zeroth order hold
dB	decibel
rpm	revolution per minute

Symbols

A/D	analog to digital conversion
$\overline{A}$	A-matrix of a similarity transformed system
A	A-matrix of a dynamic system
$\underline{A}$	lifted A-matrix

A_d	A-matrix of a discretized dynamic system
A_k	controller A-matrix
$\overline{B}$	B-matrix of a similarity transformed system
$\underline{B}$	lifted B-matrix
B_d	B-matrix of a discretized dynamic system
B_k	controller B-matrix
B_u	partition of the B-matrix related to control input
B_w	partition of the B-matrix related to disturbance input
$\mathbb{C}$	the set of all complex numbers
$\overline{C}$	C-matrix of a similarity transformed system
$\underline{C}$	lifted C-matrix
C_k	controller C-matrix
C_{yx}	sensitivity matrix relating measurement output to the state
C_{zx}	sensitivity matrix relating performance output to the state
Δt	CPU time per FLOP in second / interpolation time step
D/A	digital to analog conversion
D/C	discrete-time to continuous-time conversion
$\underline{D}$	lifted D-matrix
D_k	controller D-matrix
D_{pq}	denominator polynomial in Pade approximation
D_{yu}	D-matrix relating measurement output to control input
D_{yw}	D-matrix relating measurement output to disturbance input
D_{zu}	D-matrix relating performance output to control input
D_{zw}	D-matrix relating performance output to disturbance input
ϵ_{p2p}	percentage relative point-to-point error
ϵ_m	percentage relative mean error
ϵ_σ	percentage relative RMS error
f_s	sampling frequency
f_d	reduced sampling frequency
$G\delta_i$	impulse response due to impulse at the i^{th} input channel
H	holder
H_f	fast discretization holder

H_s	slow discretization holder
$\mathcal{H}_2$	$\mathcal{H}_2$ (Euclidean norm) control theory
$\mathcal{H}_\infty$	$\mathcal{H}_\infty$ (worst case) control theory
I	identity matrix
j	$\sqrt{-1}$
K	controller
k_n	modal gain
K_{c2d}	equivalent discrete-time controller
K_{d2c}	equivalent continuous-time controller
L	controllability grammian / lifting operation
Λ	diagonal matrix
$\mathbb{N}$	the set of all natural numbers
N_{pq}	numerator polynomial in Pade approximation
n_s	number of state variables
ω	discrete-time angular frequency [0,2π] / imaginary part of the complex frequency
Ω	continuous-time angular frequency $(-\infty, \infty)$
Ω_n	natural frequency matrix
Ω_N	bandwidth frequency
ω_{ni}	natural frequency of the i^{th} mode
ω_o	cutoff frequency of the optical controller
$\mathbb{R}$	the set of all real numbers
S	sampler
S_f	fast discretization sampler
S_s	slow discretization sampler
s	complex frequency (continuous-time)
σ	real part of the complex frequency
t_0	initial time instant
T	sampling period / nonsingular similarity transform matrix
T_{cpu}	CPU time
u	control input
V	the matrix whose columns are the eigenvectors of a matrix
w	disturbance input

$\overline{x}$	state vector of a similarity transformed system
x	state vector of a dynamic system
x_0	initial state
$x_c(t)$	continuous-time signal
$x[n]$	discrete-time signal
$\hat{x}[n]$	approximation to a signal
$X_c(j\Omega)$	Fourier transform of a continuous-time signal
$X(e^{j\omega})$	discrete-time Fourier transform of a discrete-time signal
x_H	homogeneous solution of an ODE
x_P	particular solution of an ODE
y	measurement output of a dynamic system
z	performance output of a dynamic system / complex frequency (discrete-time)
Z	damping ratio matrix
$\mathbb{Z}$	the set of all integers
ζ_i	damping ratio of the i^{th} mode
ζ_o	damping ratio of the optical controller

Chapter 1

Introduction

1.1 Research Background

Physical systems are inherently complex and one of the most popular methods to analyze them is to describe them with a mathematically tractable model based on differential equations. In the past, due to the limitation of computational devices, these models can only be of manageable dimension so the conclusions drawn from them could only be trusted with limited confidence. In the case of dynamical system design, this uncertainty results in conservative design margins that limit the performance and efficiency of the system. This is basically a passive way to solve the problem. Nowadays, with the promise of increasing computational capabilities[1], there is a desire to model the systems with better and better fidelity so that the prediction of the system behavior is more accurate and more ambitious designs can be adapted. Consequently, the models considered are getting more and more complex. For example, the model of the Space Interferometry Mission (SIM, see Figure 1-1) developed by JPL has as many as 2184 state variables.

The main problem then facing a system engineer in designing a complex dynamical system is as follows:

- It is typical that the time budgets for product/system development are fixed. Moreover, there is a trend that the budgets are getting smaller and smaller in order to adapt to the dynamics of the changing demand.

- Although the computational speed increases and cost decreases over the years, the desired model fidelity increases also and it seems to outgrow the computational power.

[1] See http://www.physics.udel.edu/wwwusers/watson/scen103/intel.html for an illustration of Moore's Law.

Figure 1-1: Space Interferometry Mission (SIM), one of the projects of the Origins Program by NASA, is a space observatory aimed at determining the positions and distances of stars several hundred times more accurately than any previous programs. It is planned to launch in 2009.

The dilemma then is: Under the constraint of a fixed time budget and computational capability, if a model with high fidelity is employed, then the number of designs explored will be limited, which means better designs might be overlooked. On the other hand, if the number of designs is increased by studying a simpler model, then concern will be raised as to whether the conclusions are reliable or not. The problem is shown graphically in Figure 1-2.

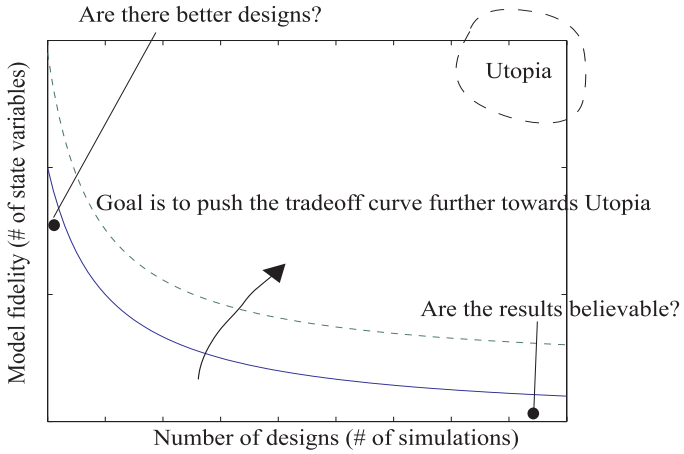


Figure 1-2: Tradeoff curve between number of designs and model fidelity. Solid line is the curve allowed by the current technique. Dashed line is the objective of this thesis. Upper right corner is the ultimate desire.

The complexity of a model can be reflected by attributes such as the number of state variables, the properties of state variables (continuous/discrete), the number of inputs and outputs, dynamic range, linearity and so on. Figure 1-3 shows the range of model dimensions

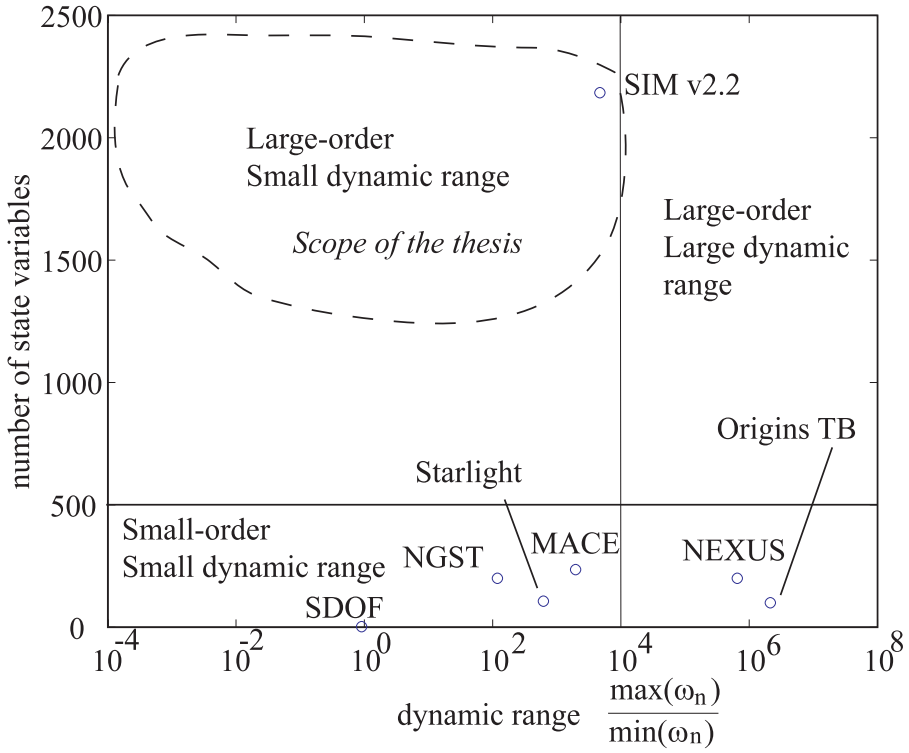


Figure 1-3: Examples of the dimensionality and dynamic range of the models of various projects. x-axis is the ratio between the highest and lowest natural frequencies of the model. y-axis is the number of state variables of the model.

and dynamics scales for selected models. Figure 1-4 shows the time response of a typical two time scale system[2] and Figure 1-5 shows the block diagram of a sampled-data control system which contains continuous-time and discrete-time state variables.

Among the models mentioned before, linear time-invariant (LTI) systems are used most extensively because they are mathematically tractable, applicable to a wide variety of problems and the theories behind them are well established. For example, $\mathcal{H}_2$, $\mathcal{H}_\infty$ and Kalman filter are among the most powerful theories for LTI systems.

Approaches to analyzing LTI systems can be categorized into two main groups, namely experimental and analytical. Experimental study of a system has the advantage that the system is actually measured. However, it suffers from the physical limitation that not

[2]See [26] for an example of simulating multiple-scaled dynamics.

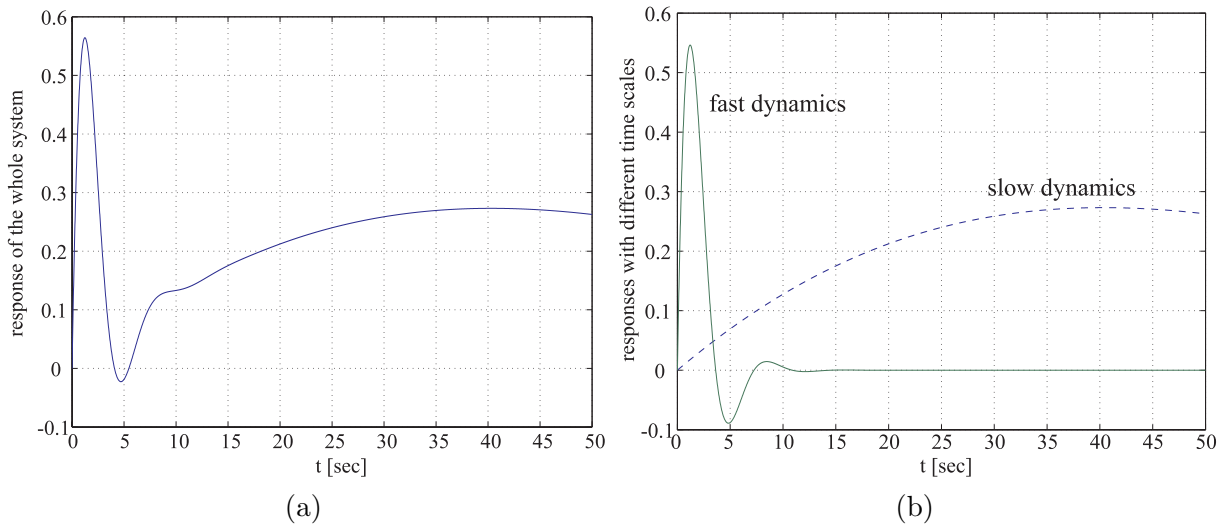


Figure 1-4: (a) Typical response of a multiple time scale system. (b) Responses of fast dynamics and slow dynamics dominate the total response in different time ranges.

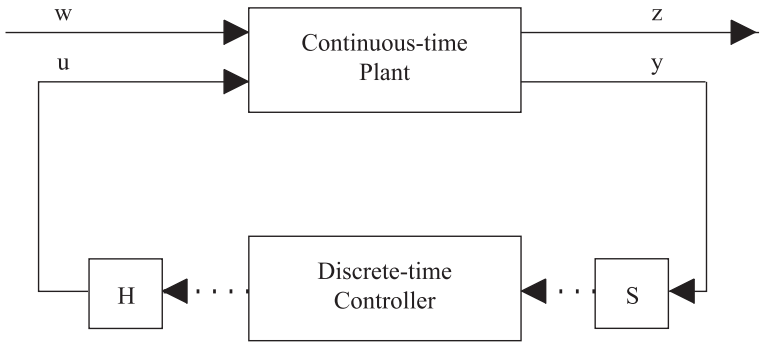


Figure 1-5: Block diagram of a sampled-data control system. S denotes the sampler and H denotes the holder.

all state variables of interest can be measured. Also, experiment is relatively expensive approach to design, thus limiting the operational range explored. The analytical method, on the other hand, is relatively cheap with the computation power available today and has the advantage that all state variables can be computed. Nevertheless, the computational result may be unrealistic unless it has been verified by an experiment. Therefore, experimental and analytical approaches are complementary. The scope of this thesis, however, will be focused on analytical study only.

The analytical study of LTI systems can also be divided into a few categories: Time domain simulation, frequency domain analysis and Lyapunov analysis. Each of these methods

are also complementary but again this thesis will focus on only one of the methods, namely, time domain simulation.

1.2 Simulating Continuous-time Linear Time-invariant Systems

The objective of this thesis is to develop a simulation tool to analyze large order continuous-time linear time-invariant systems. Time domain simulation, to be precise, is to numerically solve the differential equations governing the behavior of a model with given external input and initial or boundary conditions. The above definition includes simulations of all kinds of dynamical systems but in this thesis only LTI systems are discussed. This might seem to be rather restrictive but in fact many space systems considered are in fact LTI. They can be the original mathematical models of physical plants, but more often are obtained from linearizing nonlinear models about some specific operating point. The particular simulation problem has the following setup,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B_u u(t) + B_w w(t) \\ y(t) &= C_{yx} x(t) + D_{yu} u(t) + D_{yw} w(t) \\ z(t) &= C_{zx} x(t) + D_{zu} u(t) + D_{zw} w(t) \end{aligned} \tag{1.1}$$

where $x(t)$ is the state, $u(t)$ is the control input, $w(t)$ is the external disturbance input, $y(t)$ is the measurement output and $z(t)$ is the performance output. The matrices in (1.1) are of appropriate dimensions. The objective of a simulator is to compute $z(t)$ (and possibly $y(t)$ and $x(t)$) accurately and efficiently with external input $w(t)$ and initial conditions x_0 given.

1.2.1 The Simulation Problem

To understand where and why problems are encountered, it is necessary to understand how a computer program (e.g. MATLAB) performs the simulation. There are two common ways to solve the initial value problem which defines a simulation. One is to apply standard ordinary differential equation (ODE) solvers like Euler's method, Trapezoidal rule and Runge Kutta (e.g. `ode45.m` in MATLAB). These algorithms essentially approximate the derivative by functions evaluated at some discrete points [36]. In order to achieve accuracy and numerical stability, many function evaluations must be carried out per time step and the

time step itself cannot be too large[3]. For example, Runge Kutta 4 evaluates the derivative four times for each time step and the smallest time step is restricted by its stability function [9]. The algorithms mentioned above are very versatile in that they can solve linear and nonlinear, time varying and invariant ODE's, however, when only LTI system simulation is concerned, there is a better method. The idea of this method is to first discretize the continuous-time model by replacing it with the cascade of a holder, itself and a sampler. The holder here means zero order hold (ZOH), first order hold (FOH) or other methods to reconstruct a continuous-time signal from a discrete-time sequence. The scheme looks like Figure 1-6[4]. The equivalent discrete-time LTI system is still LTI and if zero order hold is

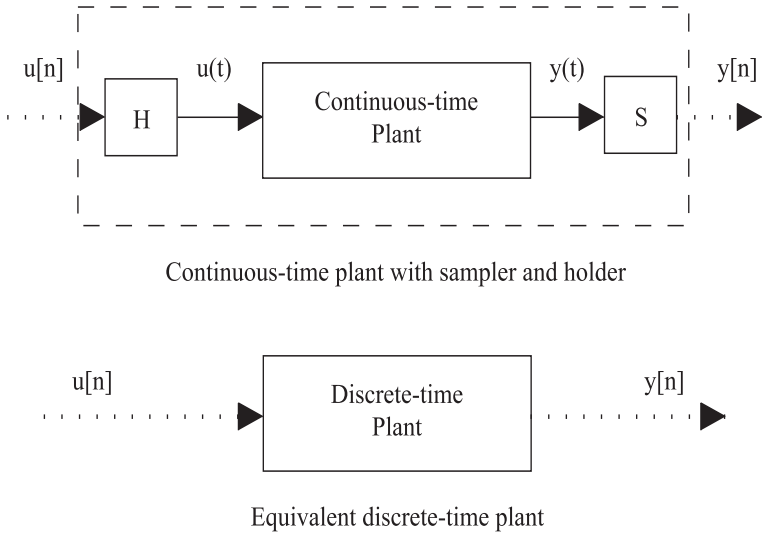


Figure 1-6: Schematic of the second method (state propagation)

used, it has the following form according to the known solution of a LTI system [4],

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned} \tag{1.2}$$

where $A_d = e^{AT}, B_d = \int_0^T e^{A\tau} B d\tau$ are the system matrices of the discrete-time system, A, B(B is the input matrix that maps all inputs to the state. Compare with 1.1), C, D are

[3]For a numerical simulation, the dynamics of the inevitable truncation error (due to finite machine precision) is governed by a difference equation whose poles are determined by the poles of the original ODE and the integration time step. For A-stable algorithms, the left half s-plane is mapped to inside the unit circle in z-plane. For example, backward Euler method. For other methods such as Forward Euler and explicit Runge Kutta, the stability can be destroyed if the time step is too long.

[4]It should be clear that in a computer simulation, there is no "true" continuous-time signal and the $u(t)$ and $y(t)$ in this figure are in fact discrete-time signals, i.e. sequences. However, software like MATLAB is smart enough to operate on them as if they were continuous-time.

the system matrices of the original continuous-time system and T is the period of sampling. The derivation of this formula will be discussed in Chapter 2. This discrete-time state propagation method has an obvious advantage over Runge Kutta in that the solution is exact and it requires much fewer arithmetic operations per time step. A simplified matrix-vector product count per time step should manifest this fact. Let's consider the case of zero input response. Then Runge Kutta 4 requires 4 matrix-vector multiplications for each major time step, while the second method (state propagation method) requires only 1 matrix-vector product from (1.2). An experiment in which the same system is simulated in the same time span by `ode45.m` (Variable step Runge Kutta[5], Dormand Prince pair [8]) and `lsim.m` (State transition method) has been conducted and the result is listed in Table 1.1. Another

Table 1.1: Some comparison between Runge Kutta method and state transition

Simulation method	RK4	State transition
Matrix-vector product per step	4	1
FLOPs in the example	88,145,484	55,509,242

advantage of the state transition scheme is that the discretized system is guaranteed to be stable if the continuous-time system is stable since if $s = \sigma + j\omega$ is a pole of the continuous-time system, then $z = e^{(\sigma+j\omega)T}$ will be the corresponding pole for the discretized system, where T is the sampling period which cannot be negative. By the argument that $\sigma < 0$, it is clear that $|e^{\sigma T}| < 1$, which implies stability in z-domain if the system is causal, which is the case. The above discussion suggests the idea that if discretization is easy to compute, the state propagation method should be used for simulating LTI systems. In fact, this is exactly what `lsim.m` in MATLAB does [14].

Now the first problem with simulating large order system is obvious. The discretization involves the computation of the matrix exponential e^{AT}, where T is the sampling period. What is actually implemented to compute e^{AT} is the Padé approximation [12],

$$e^{AT} = D_{pq}(AT)^{-1}N_{pq}(AT), \tag{1.3}$$

where p and q are the order of approximation and

$$N(AT)_{pq} = \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+q)!k!(p-k)!}(AT)^k,$$

[5]It should be pointed out that variable step ODE solvers require much less function evaluations than fixed step solvers. Variable step solver was chosen here only because fixed step solvers are not readily available in MATLAB.

and

$$D(AT)_{pq} = \sum_{k=0}^{q} \frac{(p+q-k)!q!}{(p+q)!k!(q-k)!}(-AT)^k.$$

In order to maintain accuracy of (1.3), the following condition should be satisfied:

$$\|AT\|_\infty \leq \tfrac{1}{2}.$$

Therefore, the matrix exponential is actually computed in the following way:

$$e^{AT} = (e^{AT/m})^m, \tag{1.4}$$

where m is an integer to keep the approximation accurate. From (1.4) it is clear that an indispensable part of the computation of matrix exponential is the squaring of the intermediate matrix exponential. When the A-matrix is small, evaluating its power does not hurt, but when it becomes large, this squaring becomes the bottleneck of the algorithm. Since this squaring is indispensable, the whole matrix exponential algorithm, and thus discretization becomes a problem.

The other major problem is caused by the large amount of memory required to store the intermediate and output signals. It is known that efficient matrix computation algorithms employ special processing called pipelining [12]. That is, entries of a vector are read and processed together instead of one by one. This operation is good only if the memory limit is not hit. In the case of simulating large-order systems with many sampled data sequence, the assumption of unlimited memory is not valid and swapping[6] is necessary. It is a well-known fact that swapping slows down the computation significantly because of the required read and write access times. When the dimension of the system gets even bigger and even swapping cannot provide the required memory capacity, the algorithm simply fails. This is typically the case when one tries to use `lsim.m` on MATLAB to solve large order systems. As an example of this high dimensionality problem, consider simulating the SIM v2.2 model (2184 state variables) with external input sampled at 4096Hz for 210 seconds (one set of the Magellan reaction wheel assembly disturbance input data [10]). If the history of the state is stored in a matrix of double precision, then it will require $2184 \times 4096 \times 210 \times 8 \approx 1.5 \times 10^{10}$ bytes, which is about 14 gigabytes of memory. It would be a problem if a simulator needs to store the state before computing the output. One of the examples of routines applying this

[6] Available memory on a computer consists of the physical memory (RAM) and a portion of hard disk space (virtual memory). Swapping, in principle, is the process of using the hard disk space as backup memory.

strategy is `lsim.m`. Unfortunately MATLAB on Windows has only 1.5 gigabytes of memory available even if swapping is included[7]. The two problems discussed above manifest the need to develop a simulator especially for large order LTI system simulations.

1.2.2 Assumptions and Limitations

Before discussing any solution to the problems raised in the previous subsection, it is necessary to state the assumptions about the problem.

1. A-matrix is diagonalizable. That is, for a specific $A \in \mathbb{R}^{n\times n}$, there exists a nonsingular $V \in \mathbb{R}^{n\times n}$ such that $A = V\Lambda V^{-1}$ where Λ is diagonal. The range of V is the eigenspace of A and Λ contains all the eigenvalues. A matrix that cannot be diagonalized by similarity transform is called defective. For example, an open loop system containing rigid body modes is defective. Fortunately, the stabilized system will be diagonalizable, see Chapter 3 for more details. The diagonalizability assumption allows the fast operations in the subsequent chapters.

2. Sampling rate of the known excitation is high enough such that the excitation is accurately represented by its sampled version. For example, the sampling rate should be much higher than the Nyquist rate, see Chapter 2.

3. Computational efficiency is measured in CPU times instead of FLOP count because MATLAB version 6 does not provide the floating point operations counting anymore[8]. The disadvantage of CPU time measurement is that the conclusion is load dependent and machine dependent.

1.2.3 Proposed Solution and its Philosophy

The goal of the simulator, `new_lsim`, developed in this thesis is to take advantage of the established routines like `lsim.m` without getting into their trouble with large order systems. It is not the aim of the simulator to replace established routines, but rather complementing them in such a way that the applicability of simulation design technique can be extended.

Computationally, this simulator tries to enable large order system analysis, improve the efficiency and at the same time maintain a high standard of accuracy. These requirements are to be fulfilled by the following ideas. Since the size of the problem makes it formidable,

[7] See http://www.mathworks.com/support/tech-notes/1100/1106.shtml.
[8] See http://www.mathworks.com/support/solutions/data/24987.shtml.

it seems natural to separate the work if possible. This separation is achieved through a modal decomposition (or diagonalization of the A-matrix) at the beginning of the algorithm. This diagonalization processing is a detour (or in a pessimistic sense, overhead) that is justifiable only when large order problems are concerned and the crossover above which this overhead is offset will be discussed in Chapter 2. Another important idea of this simulator is approximation in that the output is downsampled. It is the author's belief that acceptable approximation is necessary in order to achieve efficiency in a well-planned algorithm. For example, replacing the Matlab command `inv(A)*b` with `A\b` for a triangular matrix A can definitely improve efficiency and accuracy, but the former command is not well planned because it did take advantage the special structure of the matrix,

```
>> tic; inv(A)*b; toc
elapsed_time =
    3.0620
>> tic; A\b; toc
elapsed_time =
    0.0310
```

Another effort made in this simulator is the treatment of vectors and matrices especially for large systems. This is necessary because memory flow and storage is also a decisive factor of algorithm efficiency [12]. Although improvements at the "computer science" level are beyond scope of this thesis, care has been taken to avoid that the matrices are getting too large. For example, simulating subproblems can immediately relieve the problem of not having enough memory. The underlying theme of these treatments is to become a better user without becoming a software developer.

1.3 Previous Work

Existing runtime reduction strategies can be categorized into three groups.

1. **Improving hardware capability**. For example, due to very large scale integration (VLSI) technology, the number of transistors (computation speed) on a single chip is increasing over years according to Moore's Law. In addition to that, the cost of relevant components such as random access memory (RAM) and hard disks is decreasing due to new technologies. All these contribute to the ever increasing computation capability. However, the growth pace of device is always slower than that of the desire.

In addition to that, due to limited financial budgets of design, this improved hardware capability method is not preferable since it induces extra cost.

2. **Model reduction**. The idea of model reduction of a model is to retain only a few modes which are significant to the input/output relation. The reduction procedure can usually be divided into two parts: A similarity transform (for state space system) is applied to the model so that the "significance" of the modes is revealed. Then some kind of criterion (e.g. $\mathcal{H}_2$, $\mathcal{H}_\infty$ or Hankel norm) is chosen to provide the justification to throw away the "unnecessary" modes. Thus a reduced model with much lower order than the original one is obtained. This method is very powerful in terms of runtime reduction, and is especially important to control system design, implementation and to the understanding of the principal physical phenomena of a complex dynamical system, where a simplified model is required. The idea of model reduction began with the seminal papers such as Moore [24], Laub [21] and Laub [22]. Papers especially dedicated to space-borne structure applications are Gregory [15], Skelton [28] and Uebelhart [35]. More recent development of model reduction schemes and algorithms can be found in papers such as Myles et al [31], where an approximate subspace iteration procedure was described to find the reduced model efficiently by exploiting the sparsity structure of the matrices in structural dynamics problems, Gawronski [11] in which a scaling of modal gain is employed to allow efficient balancing, Willcox and Peraire [37] proposes a promising reduction method based on proper orthogonal decomposition and Beran and Silva [1] gave an overview of some promising model reduction methods.

   Although model reduction methods work, it is not always desirable for some reasons. Engineers have "pride" in high-fidelity models and they do not want the models reduced (e.g. JPL SIM modelling team explicitly states that the 2184-state variable model has been reduced and it should not be further reduced.). Moreover, reduction is always a compromise between model complexity and accuracy and it is somewhat arbitrary. For example, the chosen criteria such as $\mathcal{H}_2$, $\mathcal{H}_\infty$ and Hankel norm are just the distance between the original and approximated model in some sense. The error of reduction is still an open problem and further investigations are needed to understand it. This uncertainty in error is especially undesirable in designs where high precision

is required and a high price will be paid if the design fails (e.g. SIM, NGST, TPF). For the reasons above, a complementary design method is needed and this gives rise to the third runtime reduction method, namely, decoupling and parallelization.

3. **Decoupling and parallelization**. The method of parallelization is basically a divide and conquer method. Its principle is to divide a large problem into a set of smaller problems so that they can be solved concurrently. This will not spare computation effort but it definitely reduces computation time. In cases where multiple processors are not available, this decoupling and parallelization method still has its value in improving computational efficiency because it allows the problem size to be reduced when the original problem is considered too large by the processor. For example, the QR based eigenvalue/eigenvector solver is being challenged by the new idea of a divide and conquer algorithm [34]. Decoupling and parallelization has the advantage over model reduction in that no arbitrary decision about the acceptable model reduction level is required and the inherent sparsity of some problems (e.g. structural dynamics problems) can be exploited. Papers applying parallelization methods for simulation are, for example, Christopher et al [2] and Dong and Zhong [7]. However, most of these existing papers seem to deal only with partial differential equation (PDE) problems and the solvers are always Runge Kutta based instead of using the state transition idea discussed. More space system engineering related works include de Weck [6], where an efficient Lyapunov equation solver via decoupling is proposed and Maghami and Kenny [23] exploited the sparsity structure to reduce the computational burden of finding the frequency response of a system. However, these papers do not address the issue of time domain simulation as is the purpose of this thesis. Ha [16] described a method for sampled-data control system simulation but it didn't discuss the problem of large-order system simulation either.

In the wake of the lack of work on simulating large-order structural dynamics ODE problems with the decoupling and parallelization[9] method, a MATLAB based simulator[10] , `new_lsim.m` is developed to fill the gap.

[9]Parallel computing is not employed in this thesis, but the idea developed readily lends itself to the implementation of parallel computation if allowed by the computation resources.

[10]It is not necessary that the code be written in MATLAB, but MATLAB seems to be the first choice for a quick algorithm development platform because it is powerful and easy to use.

1.4 Overview

The organization of the subsequent chapters is as follows. Chapter 2 discusses the issues of open loop simulation. First the exact problem will be defined, then a flow graph of the operations will be given to show the framework of the simulator. Then the motivation and mathematical basis will be explained. After that, details on implementation issues will be addressed. This discussion will be followed by examples with applications. Finally accuracy issue will be studied and improvements will be recommended.

Chapter 3 will focus on closed loop (with continuous-time plant and discrete-time controller) simulation and the structure of this chapter will be similar to that of chapter 2. The expanded problem will first be defined, operations will be mentioned in general, followed by details and rationale of each operation. Simulation results of a sample problem will then be given and finally error analysis and improvement will be discussed.

In chapter 4 the SIM model will be analyzed by the simulator. This will serve as an example of a real world application. Numerical result will be compared with the existing MATLAB routine `lsim.m`. In particular, the chapter will leverage the new simulation algorithm for steady-state and transient performance assessment as well as a finite difference based approximation of the Jacobian matrix.

Chapter 5 will contain conclusions and recommendations for future work.

Finally, the roadmap of the thesis is given in Figure 1-7.

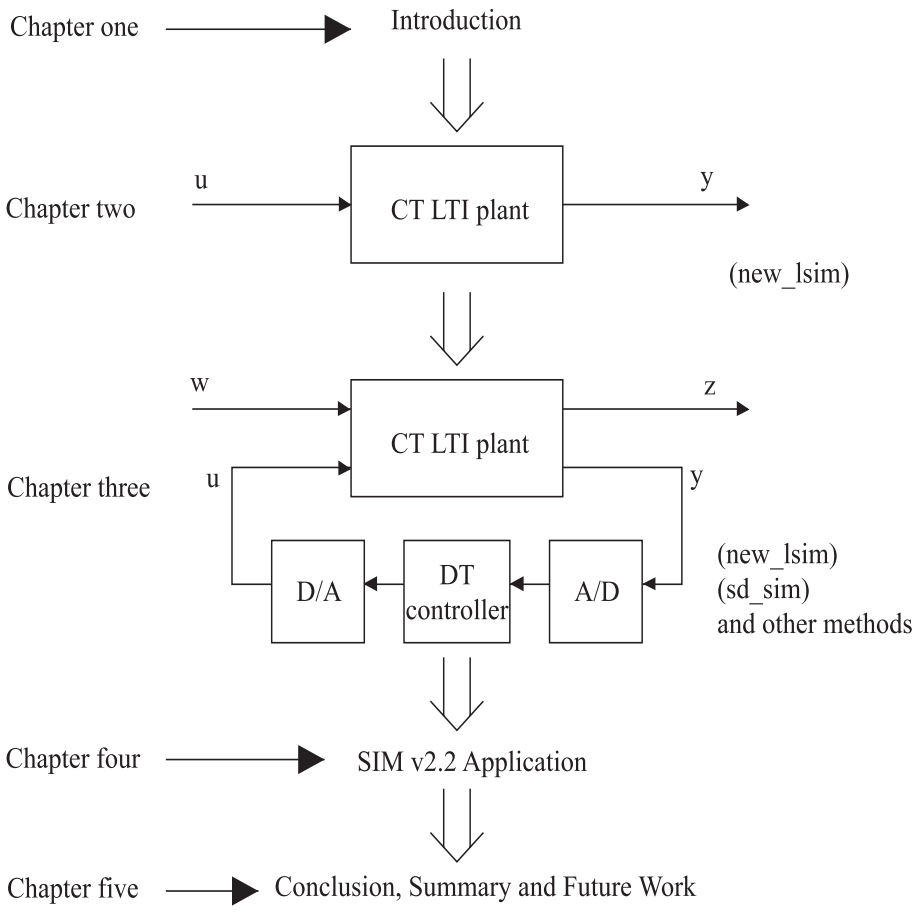


Figure 1-7: Roadmap of the thesis.

Chapter 2

Open Loop System Simulation

In this chapter, the standard from of LTI systems is of interest. Given an LTI system,

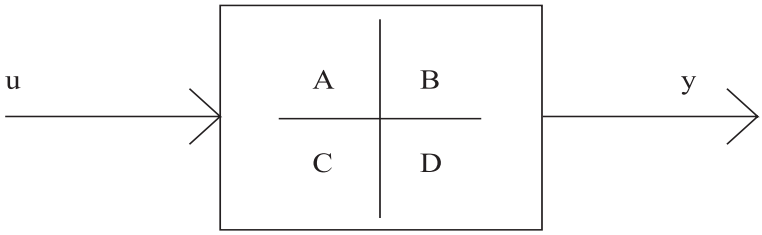


Figure 2-1: Block diagram of an open loop system in state space form

with external input $u(t)$ and initial conditions x_0, the simulator calculates the output $y(t)$. The block diagram represents an "open loop" system in that the feedback loop of a complete feedback control system is removed. The only block shown here is the plant, which can be unstable or neutrally stable[1] by itself. Whereas a "closed loop" system is a system with one or more feedback control loops and the overall system should be stable and have satisfactory performance such as good transient response, relative stability and robustness. Although the performance of a feedback control system is usually the primary concern, the method to simulate an open loop system has interest in its own right for several reasons. First, the state space model figure 2-1 is generic since any feedback control system can be represented by an autonomous system as an open loop system. Secondly, as will be discussed in Chapter 3, an open loop system simulation routine is an essential part of a closed loop system simulator. Finally, sometimes it is interesting to see how the plant behaves without any stabilization.

[1]Neutral stability implies that there are poles, but only single order poles, on the imaginary axis in addition to poles in left half s-plane for a continuous-time system.

2.1 Some Features of the Problem of Interest

As discussed in Chapter 1, the aim of the simulator, `new_lsim.m`, is not to replace the established routines like `lsim.m`, but rather to complement them in case of large-order system simulations. Therefore, the problem should have significant "size" for `new_lsim.m` to show its value. Here the "size" can mean the dimension of the state vector (n_s), which is also reflected by the dimension of the A-matrix, the number of samples to be processed (length of $u(t)$), which is decided by the sampling rate and time span of interest and the number of input and output channels in case of MIMO system. Whether the A-matrix is big enough to warrant using `new_lsim.m` is not easy to justify but as a rule of thumb, when the cost of computing the diagonalization (`ss2mod71.m`) is smaller than direct simulation, the method of `new_lsim.m` is justified since the simulation cost of `new_lsim.m` depends only linearly on the the number of state variables (n_s) and the details will be given in Section 2.5. The following example should make this point clear, see Table 2.1. The results are obtained from simulating randomly generated state space models (2 inputs and 2 outputs) with different n_s, with randomly generated input of fixed length (32768 points) and randomly generated initial values. Figure 2-2 is a graphical representation of the result.

Table 2.1: Computation time (in seconds) versus number of state variables

Number of state variables	5	10	20	50	70	100
`lsim.m`	0.1100	0.1720	0.2810	0.9220	1.4380	2.3600
`new_lsim.m` and `ss2mod71.m`	0.2030	0.1880	0.2190	0.2810	0.3280	0.4530
relative error ($\times 10^{-9}$ %)	0.0412	0.3611	0.0915	0.0673	0.7018	0.1050
Number of state variables	150	200	300	400	500	600
`lsim.m`	10.7660	23.5310	75.9840	207.9850	366.8600	559.2810
`new_lsim.m` and `ss2mod71.m`	0.7030	1.2500	3.0160	5.9220	11.0470	16.3440
relative error ($\times 10^{-9}$ %)	0.3281	0.7234	0.1218	0.2652	0.1151	0.6464

The structure of the A-matrix is also important. For example, if it is already in diagonal form or can be transformed to diagonal form by simple manipulation such as indexing, then the diagonalization step can be omitted. Figure 2-3 (b) is the sparsity plot of the A-matrix of SIM model v2.2 and it is the latter form mentioned above. The structure of the A-matrix

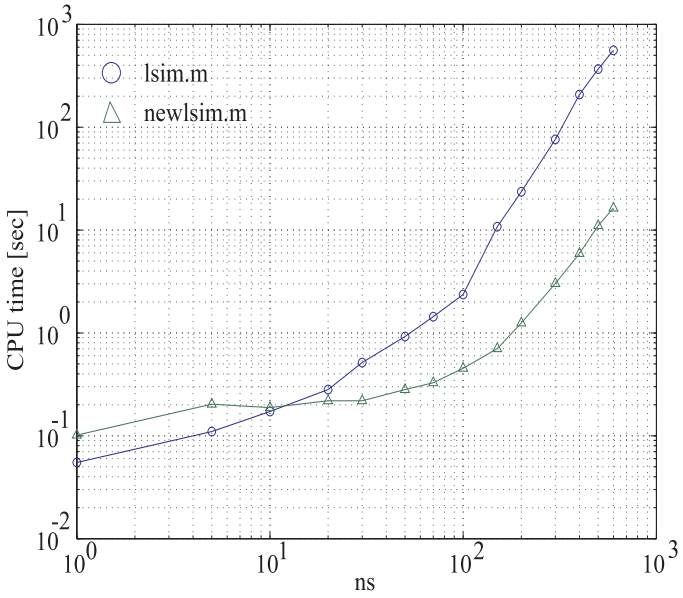


Figure 2-2: Logarithmic plot of CPU times by `lsim.m` and `new_lsim.m` versus number of state variables. Circle: `lsim.m` time. Triangle: `new_lsim.m` time.

has the following physical meaning.

$$A = \begin{bmatrix} 0 & I \\ -\Omega_n^2 & -2Z\Omega_n \end{bmatrix} \tag{2.1}$$

where,

$$\Omega_n = \begin{bmatrix} \omega_{n1} & 0 & \cdots & 0 \\ 0 & \omega_{n2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \omega_{nN} \end{bmatrix} \quad Z = \begin{bmatrix} \zeta_1 & 0 & \cdots & 0 \\ 0 & \zeta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \zeta_N \end{bmatrix} \tag{2.2}$$

where Ω_n is the modal frequency matrix and Z is the modal damping ratio matrix. However, generally the A-matrix of a dynamical system is very dense and has the sparsity plot like Figure 2-3 (a). In this case, diagonalization will be necessary, as will be discussed in detail later this chapter.

2.2 Flow Chart and Operations Introduction

In this section the flow chart of `new_lsim.m` is given to illustrate the framework of this simulator. See Figure 2-4.

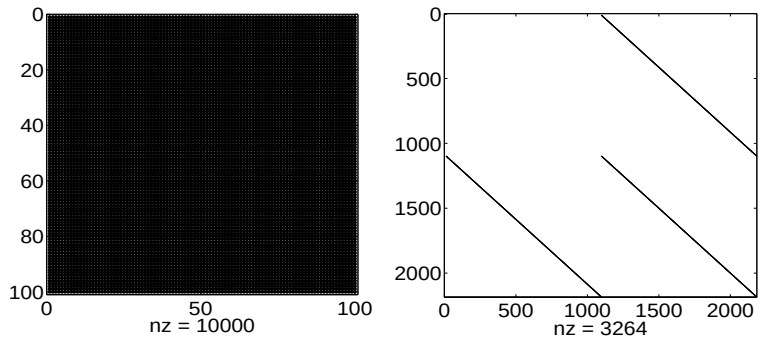


Figure 2-3: (a) Sparsity plot of a general A-matrix. (b) Sparsity plot of the A-matrix of SIM v2.2 in 2^{nd} order modal form.

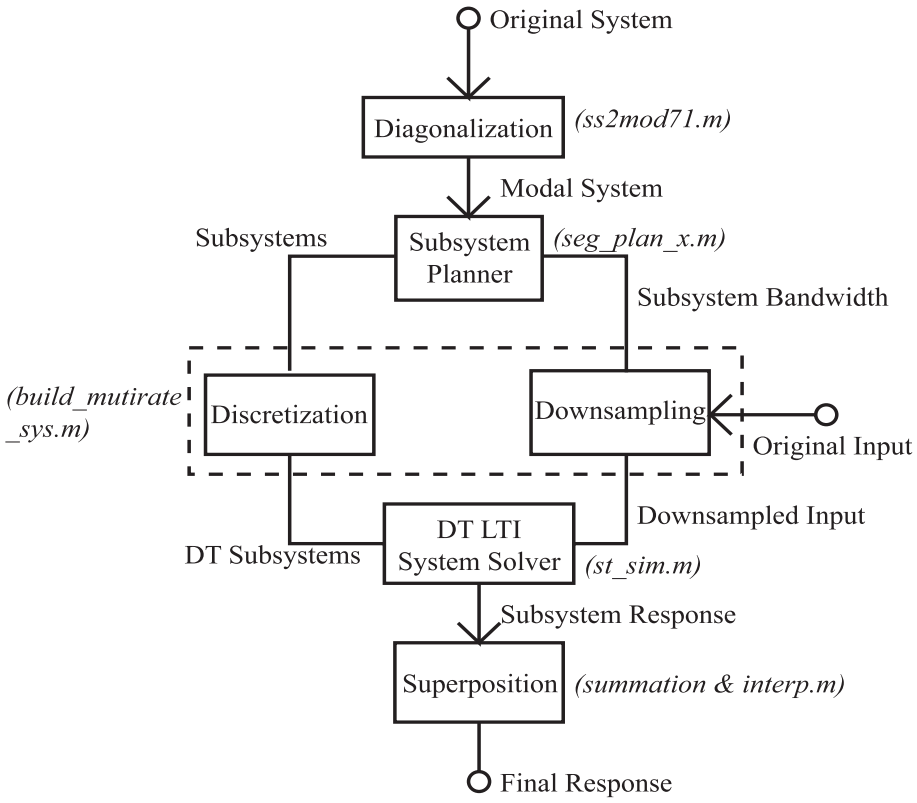


Figure 2-4: Flow chart of `new_lsim.m`

In this flow chart, the circles correspond to conventional data such as the original system (ABCD matrices), and the external input and output. Each block represents an operation or process with the actual MATLAB implementation beside it (The name of the m-file is in italic font). A brief discussion of all the operations is given here and their rationale and

details will be studied in Section 2.3 and 2.4 respectively. If the original system is diagonal or can be transformed to a diagonal form simply by indexing (e.g. Figure 2-3 (b)), then the first block (diagonalization) can be omitted, otherwise diagonalization, which essentially is a similarity transform [38], is necessary. The issue regarding diagonalization will be discussed in more detail in Appendix A. Now let's come back to the issue of similarity transform. Let $x(t) = T\bar{x}(t)$, where T is a nonsingular square matrix and substitute x into the state equation,

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.3}$$

$$\begin{cases} T\dot{\bar{x}}(t) = AT\bar{x}(t) + Bu(t) \\ y(t) = CT\bar{x}(t) + Du(t) \end{cases} \tag{2.4}$$

$$\begin{cases} \dot{\bar{x}}(t) = T^{-1}AT\bar{x}(t) + T^{-1}Bu(t) \\ y(t) = CT\bar{x}(t) + Du(t) \end{cases} \tag{2.5}$$

$$\begin{cases} \dot{\bar{x}}(t) = \bar{A}\bar{x}(t) + \bar{B}u(t) \\ y(t) = \bar{C}\bar{x}(t) + Du(t) \end{cases} \tag{2.6}$$

and $\bar{A}$ is diagonal. It should be noted that $\bar{A}$ is not generally real even though A is real. However, another similarity transform can be employed to obtain a real 2×2 block diagonal A-matrix. See Appendix A for more detail. In the following argument, it is assumed that the transformed A-matrix is in block-diagonal form instead of strictly diagonal form. The resultant system by a similarity transform is called a modal system in that the components of the state vector now represent coordinates of the modes of the system. The m-file that implements this diagonalization is `ss2mod71.m` (see Appendix A) and it is a subroutine that is used by `new_lsim.m`. Another functionality of this m-file is to sort the modes in terms of their natural frequencies, which are closely related to the bandwidth of the subsystems that include these modes. Then the modal system is fed to the subsystem planner block, which comprises a simple logic to make the decision on how to group the modes into subsystems and assign an appropriate sampling rate to each of the subsystems. The flow chart of the implementation of the subsystem planner is given in Figure 2-18. Subsystems formed in this block are then fed to the discretization block in which the discrete-time subsystems are calculated using the MATLAB routine `c2d.m` (i.e. continuous-time to discrete-time conversion [14]). Parallel to the discretization block is the downsampling block. Downsam-

pling processing is to reduce the number samples in such a way that the resultant result is acceptable. Discretization and downsampling blocks are enclosed by the dot line rectangle because they are the two functions of a single m-file, `build_multirate_sys.m`. With the discrete-time subsystems and external input with corresponding sampling rate, the responses of these subsystems can be computed (by `st_sim.m`) and then superposed to form the total response as though it were the output of the original system[2].

2.3 Motivation and Mathematical Basis of the Operations

In this section, the motivations and mathematical basis of the proposed operations will be discussed.

- As discussed in Chapter 1, it is advantageous to transform the original continuous time system

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.7}$$

to its discrete-time counterpart

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n], \end{aligned} \tag{2.8}$$

where $A_d = e^{AT}$, $B_d = \int_0^T e^{A\tau} B d\tau$ and T is the sampling period. Let's derive these two important matrices here. The derivation is accomplished by finding the analytical form of the solution of (2.7) and it is well-known that (2.7) can be solved in two steps [17]. The derivation here is based on [30]. Let's begin with the homogeneous equation

$$\dot{x}(t) = Ax(t). \tag{2.9}$$

Assume the solution is sufficiently smooth such that a series expansion of the homogeneous solution is

$$x_H(t) = A_0 + A_1(t - t_0) + A_2(t - t_0)^2 + \cdots \tag{2.10}$$

where $A_0, A_1, \ldots$ can be matrices of appropriate dimensions and t_0 is the initial time. It is obvious that $A_0 = x_0$ where x_0 is the initial condition by setting $t = t_0$. Then

[2]This principle of superposition is a direct consequence of the linearity property of the LTI systems of interest.

differentiate (2.10) with respect to t, substitute it into (2.9) and set $t = t_0$, we get $A_1 = Ax_0$. In similar fashion, differentiate (2.9) and (2.10) and plug it back, we finally come up with the expression

$$x_H = [I + A(t - t_0) + \frac{A^2(t-t_0)^2}{2} + \frac{A^3(t-t_0)^3}{3!} + \cdots]x(t_0) \tag{2.11}$$

and the expression inside brackets is defined as $e^{A(t-t_0)}$ and is called the matrix exponential of $A(t - t_0)$. It has the significant meaning of state transition, namely if $x(t_0)$ is given, then $x(t) = e^{A(t-t_0)}x(t_0)\ \forall\ t \in \mathbb{R}$ is known[3]. With the definition above, (2.11) simplifies to

$$x_H = e^{A(t-t_0)}x(t_0). \tag{2.12}$$

For the propagation of state in one time step T, we get $A_d = e^{AT}$. An interesting property of this function is that $e^{AT}e^{-AT} = I$. This is obvious if one regards the matrix exponential as state transition. Another property of the matrix exponential is $\frac{\partial}{\partial T}e^{AT} = Ae^{AT}$. Just plug x_H back to (2.9) to see why.

Now let's turn to the second part of the solution of (2.9), namely the particular solution x_P. It is the solution of (2.7) with initial condition x_0 equal to zero. Variation of parameter [17] is used to solve this problem. Assume

$$x_P(t) = e^{A(t-t_0)}v(t), \tag{2.13}$$

where $v(t)$ is a vector of variable parameters to be determined. Substitute (2.13) into the first equation of (2.7) to obtain

$$Ae^{A(t-t_0)}v + e^{A(t-t_0)}\dot{v} = Ae^{A(t-t_0)}v + Bu. \tag{2.14}$$

From the above equation and the inverse property of matrix exponential, we obtain

$$\dot{v} = e^{-A(t-t_0)}Bu(t) \tag{2.15}$$

and if the assumption $u(t) = 0\ \forall\ t < t_0$ is valid[4], then

$$v(t) = \int_{t_0}^{t} e^{-A(\tau-t_0)}Bu(\tau)d\tau. \tag{2.16}$$

[3]It is common to regard that $t > t_0$ but the expression is true for $t < t_0$ too, since state transition can go backward as well as forward.

[4]In case of control system analysis, $u(t)$ is the control input and is usually zero before the initial time concerned.

Plug the above expression back to (2.13), we obtain

$$x_P = \int_{t_0}^{t} e^{A(t-\tau)} B u(\tau) d\tau. \tag{2.17}$$

Combine (2.11) and (2.17) we finally obtain the analytical form of the solution of the first equation of (2.7)

$$x(t) = \underbrace{e^{A(t-t_0)} x(t_0)}_{x_H} + \underbrace{\int_{t_0}^{t} e^{A(t-\tau)} B u(\tau) d\tau}_{x_p} \tag{2.18}$$

The A_d in (2.8) is obvious by setting $t - t_0 = T$ in (2.18). In order to obtain B_d it is necessary to make a further assumption. For the sake of simplicity, it is assumed that $u(t)$ is a piecewise constant function and that the switch points are at $t = nT$ where $n \in \mathbb{Z}$. Then the integral term of (2.18) becomes $(\int_{t_0}^{t} e^{A(t-\tau)} B d\tau) u(nT) \ \forall \ t \in [nT, (n+1)T)$. Now consider (2.18) for $t_0 = nT$ where T is the time step and $n \in \mathbb{Z}$, it becomes

$$x(nT + T) = e^{AT} x(nT) + (\int_{nT}^{nT+T} e^{A\left((n+1)T-\tau\right)} B d\tau) u(nT). \tag{2.19}$$

By introducing the change of variables $\tau = (n+1)T - \eta$, then (2.19) becomes

$$x(nT + T) = e^{AT} x(nT) + (\int_{0}^{T} e^{A\eta} B d\eta) u(nT). \tag{2.20}$$

If discrete-time signal format is used and the above form is compared with (2.8), then

$$A_d = e^{AT} \qquad B_d = \int_{0}^{T} e^{A\tau} B d\tau \tag{2.21}$$

as desired.

- The assumption of a diagonalizable A-matrix makes it possible to transform (2.3) to (2.6). But why bother spending computation resources on this overhead? The answer is, that diagonalization serves as the preparation for fast computation schemes that follow. The fundamental concept of this overhead is that it decouples the problem and results in the following benefits. The first benefit is fast discretization. Discretization computes two matrices $A_d = e^{AT}$ and $B_d = \int_0^T e^{A\tau} B d\tau$ and these computations are in fact the computation of matrix exponentials. If A is diagonal, then it is readily observed that $A_d = e^{AT} = I + A(t - t_0) + \frac{A^2(t-t_0)^2}{2} + \frac{A^3(t-t_0)^3}{3!} + \cdots$ is diagonal by noting the fact that the matrix product of two diagonal matrices is still diagonal. As

a matter of fact, the matrix exponential of a diagonal matrix is a diagonal matrix with entries being the exponentials of those of the original matrix. That is,

$$\exp\left(\begin{bmatrix} a_{11}t & 0 & \cdots & 0 \\ 0 & a_{22}t & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn}t \end{bmatrix}\right) = \begin{bmatrix} e^{a_{11}t} & 0 & \cdots & 0 \\ 0 & e^{a_{22}t} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & e^{a_{nn}t} \end{bmatrix}. \tag{2.22}$$

This immediately relieves the burden of calculating $A_d = e^{AT}$ since scalar exponentials are cheap. It should be noted that the "entries" a_{ij} in (2.22) can be blocks in terms of block diagonal matrix exponential. As a comparison, the number of multiplications for squaring a dense matrix is $\mathrm{O}(n^3)$ but that of the sparse case here is only $\mathrm{O}(n)$, where n is the number of columns (rows) of the matrix. The following little experiment on MATLAB shows the difference.

```
>> A = diag(randn(1000,1));                % create a random matrix
>> tic; Aexp = expm(A); toc                % dense matrix exponential
elapsed_time =                             % computation time
   27.1090
>> tic; Aexp1 = diag(exp(diag(A))); toc     % diagonal matrix exponential
elapsed_time =                             % computation time
    0.0310
>> norm(Aexp - Aexp1)/length(Aexp)^2        % error checking
ans =
  3.7303e-020
```

This is the basic concept of the fast discretization `new_lsim.m` realizes but by no means the only thing `new_lsim.m` implements because of the need to compute the other matrix, namely B_d. The computation of B_d seems to be more complicated but insights can be gained by studying the definition of B_d (2.21). Let's evaluate the integral $\int_0^T e^{A\tau}Bd\tau$ in its expansion term by term,

$$\begin{aligned} \left[\int_0^T (I + A\tau + \frac{A^2\tau^2}{2} + \cdots)d\tau\right] B &= (T + \frac{AT^2}{2!} + \frac{A^2T^3}{3!} + \cdots)B \\ &= (\sum_{k=0}^{\infty} \frac{A^kT^{k+1}}{(k+1)!})B, \end{aligned}$$

and this becomes

$$B_d = (\sum_{k=0}^{\infty} \tfrac{A^kT^k}{(k+1)!})TB.$$

Note that the summation in parenthesis is a diagonal matrix since A is diagonal. Now it is clear how the diagonalization of A paves the way to efficient discretization.

The above derivation is based on the assumption of zero order hold (ZOH) but the idea of decoupling can still apply to different discretization schemes like first order hold (FOH), see Appendix B.

The other benefit of decoupling the problem is the flexibility to organize the simulation. This is true because diagonalization breaks down the original problem into a group of independent 1st order and 2nd order problems (i.e. problems containing only one real or complex mode. See Figure 2-5). On the other hand, the linearity property states that the linear combination of these modal responses is equal to the response of system with all the modes combined. These two facts altogether constitute the underlying reason for the success of the modal decomposition scheme. This scheme allows the simulator to create fictitious subsystems by grouping the modes. Figure 2-6 illustrates this idea.

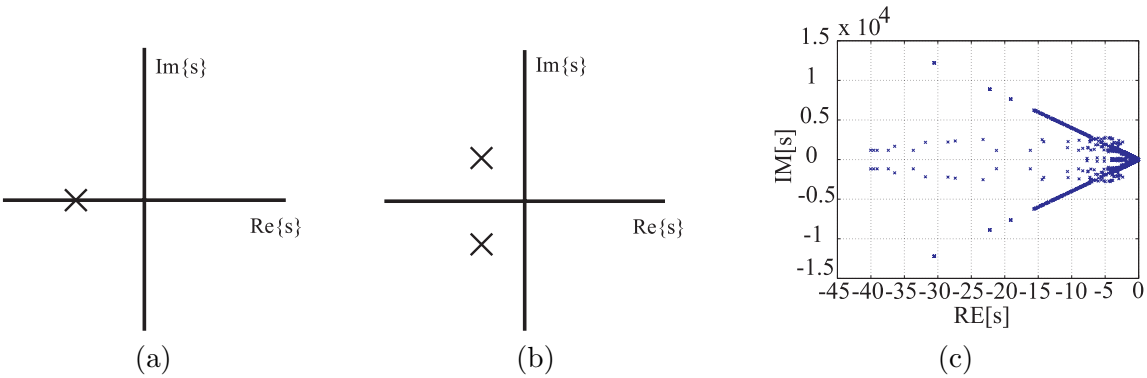


Figure 2-5: (a) Pole diagram of the real mode system. (b) Pole zero diagram of a complex mode system. (c) Pole location of the 2184-state variable SIM model v2.2.

A direct result of the modal decomposition scheme is that the problem size is reduced, and so is the memory requirement.

Diagonalization offers other benefits in terms of computation. For example, the diagonal structure essentially reduces the number of multiplications per step of state transition from $O(n_s^2)$ to $O(n_s)$, which is the main reason why `new_lsim.m` is so much faster than `lsim.m` as in Table 2.1.

- The independence of subsystems allows independent treatments of each of them. One

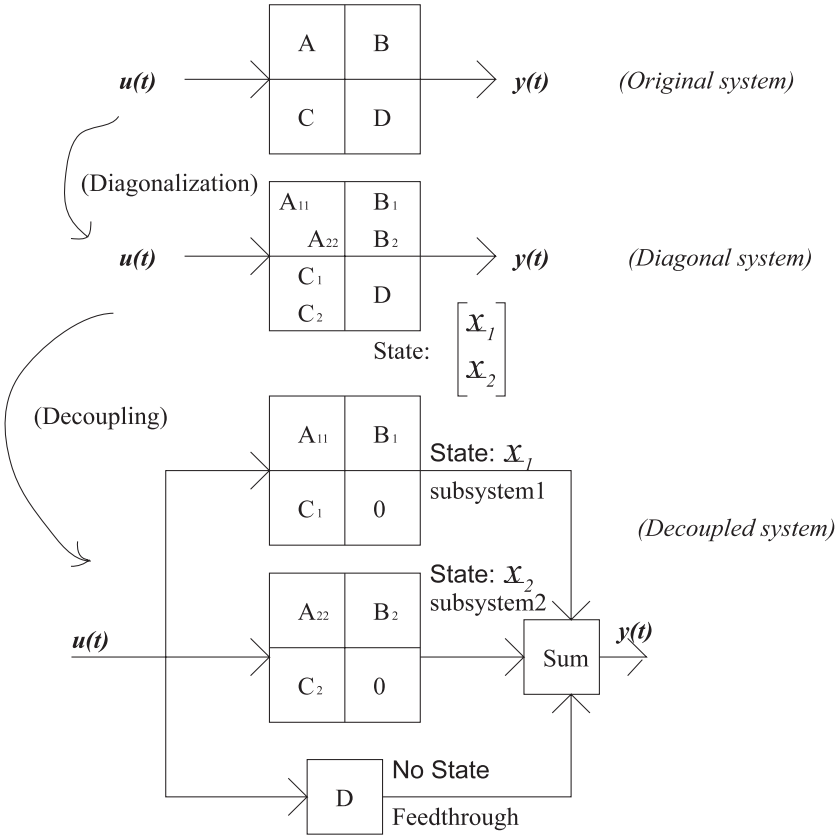


Figure 2-6: Top: Original plant. Middle: Diagonal system. Bottom: Modal decomposition by fictitious subsystems

of the most interesting aspects of these treatments is the sampling rate in that multiple sampling rates is one of the tricks to facilitate the simulation of a large-order system. In order to understand the benefits and drawbacks of this approach, it is necessary to understand the relationship between a continuous-time signal $x_c(t)$ and its sampled version $x[n]$ in the time and frequency domains. In the time domain, the relationship[5] is $x[n] = x_c(nT)$, where T is the sampling period. In the frequency domain, these signals are related by $\mathrm{X}(e^{j\omega}) = \frac{1}{T}\sum_{n=-\infty}^{\infty} \mathrm{X}_c(j(\frac{\omega}{T} - n\frac{2\pi}{T}))$, where $\mathrm{X}_c(j\Omega)$ is the Fourier transform (FT) of $x_c(t)$, $\mathrm{X}(e^{j\omega})$ is the discrete-time Fourier transform (DTFT) of $x[n]$ and T is the sampling period. The above expression can be represented as in Figure 2-8[6]. That is, the discrete-time Fourier transform of the sampled signal is the superposition of infinitely many replications of the amplitude scaled version of the

[5]Periodic sampling is assumed, i.e. the sampling period T is constant for all time steps.

[6]The assumption of this figure is that the sampling rate is high enough. For example, the sampling rate is higher than twice the bandwidth of the signal $x_c(t)$.

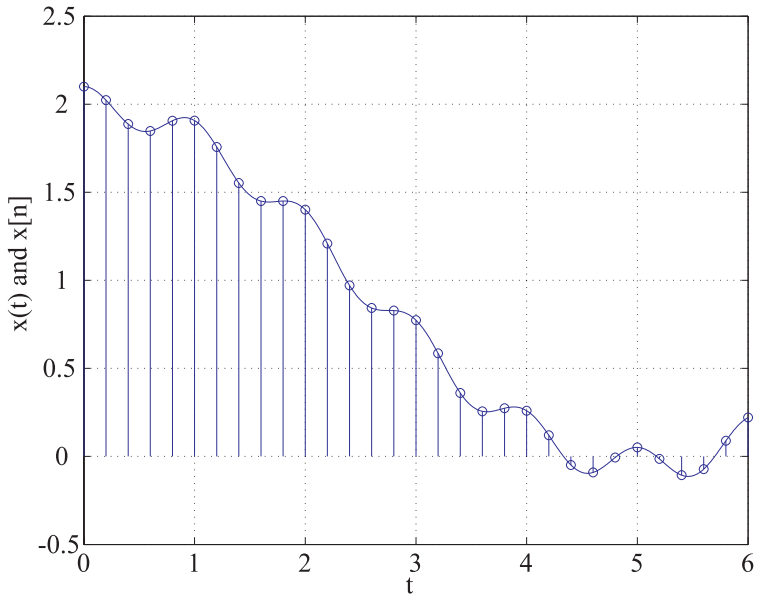


Figure 2-7: A continuous-time signal and its sampled version.

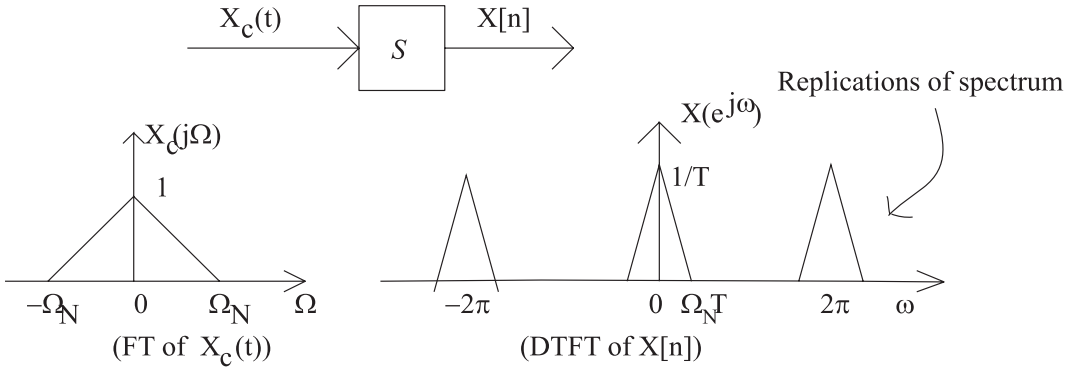


Figure 2-8: Relationship between the FT of a CT signal and DTFT of a DT signal

Fourier transform of the continuous-time signal, with the frequency axis relationship $\omega = \Omega T$. Figure 2-8 explains this graphically. A immediate result is that as sampling period changes, the relative position of the replications of spectra change. The bottom line is that the replications cannot overlap. This issue and the corresponding remedy will be discussed in Section 2.4.

- The fact that the subsystems are bandlimited is what actually allows fast simulation. Theoretically, a system is bandlimited if $\mathrm{H}_c(\mathrm{j}\Omega) = 0 \ \forall \ |\Omega| > \Omega_N$, where $\mathrm{H}_c(\mathrm{j}\Omega)$ is the Fourier transform of the impulse response of a system, Ω is angular frequency

in rad/sec and Ω_N is called the bandwidth. In other words, the impulse response of the system contains no frequency content above Ω_N. This is an idealized concept and practically Ω_N is defined to be some frequency beyond which $|\,H_c(j\Omega)|$ is less than some given value. See Figure 2-9 for an example. In this example, magnitude responses at frequencies above 3×10^3 rad/sec are considered zero since they will be no greater than 0.1% of the DC gain. With the concept of a bandlimited system, signal components at frequencies beyond bandwidth are considered as unimportant and can be neglected. This idea is the rationale behind downsampling. In the subsystem planner subroutine, the combined information on system frequency response and external input energy is exploited to obtain a practical bandwidth allowed by a given error tolerance.

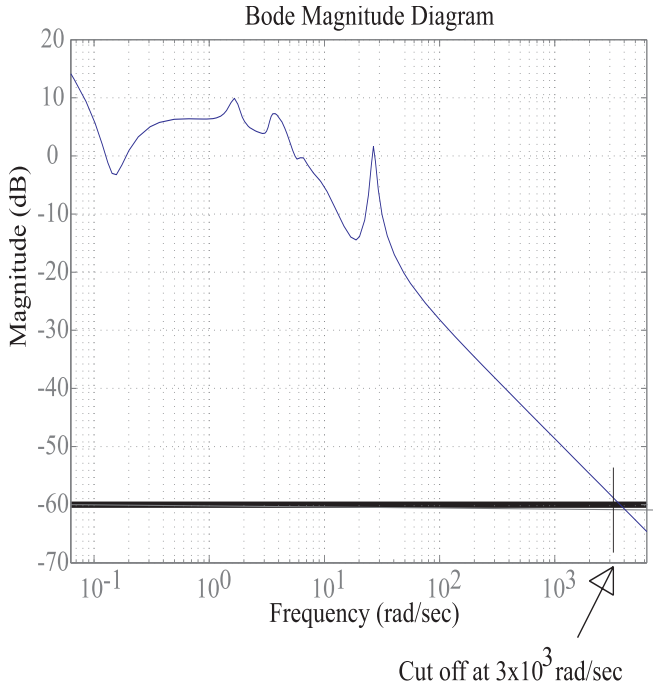


Figure 2-9: Bode diagram of a typical dynamical system and its bandwidth. The bandwidth is arbitrarily set here but in the subsystem planner subroutine `seg_plan_x.m` the bandwidth is affected by the desired error level.

This section summarized the foundations of the manipulations shown in the framework of Figure 2-4 in Section 2.2 and now we are in the position to detail the considerations of

each of the operations.

2.4 Details on Operations and Considerations

In this section the implementation details will be discussed and the sequence of these operations will follow that of the flow chart Figure 2-4 in Section 2.2.

2.4.1 Modal Decomposition

Before the simulator can simulate the system, it is necessary to make sure that the system is in the required modal form, as discussed in Section 2.3. The script `ss2mod71.m` is written for this purpose. It essentially implements the similarity transform provided by the MATLAB command `canon.m` (Canonical state-space realizations). In addition to that, `ss2mod71.m` sorts the modes of the system in the following way. A subsystem of real modes is followed by a subsystem of complex modes. The real modes are sorted with ascending magnitudes of eigenvalues[7] and the complex modes are sorted with ascending natural frequencies[8]. This sorting makes it easier for the simulator to form the fictitious subsystems and assign the "appropriate" sampling rates since the subsystems are bandlimited. A typical sparsity plot of the resultant diagonal A-matrix is like Figure 2-10. Finally, `ss2mod71.m` normalizes the

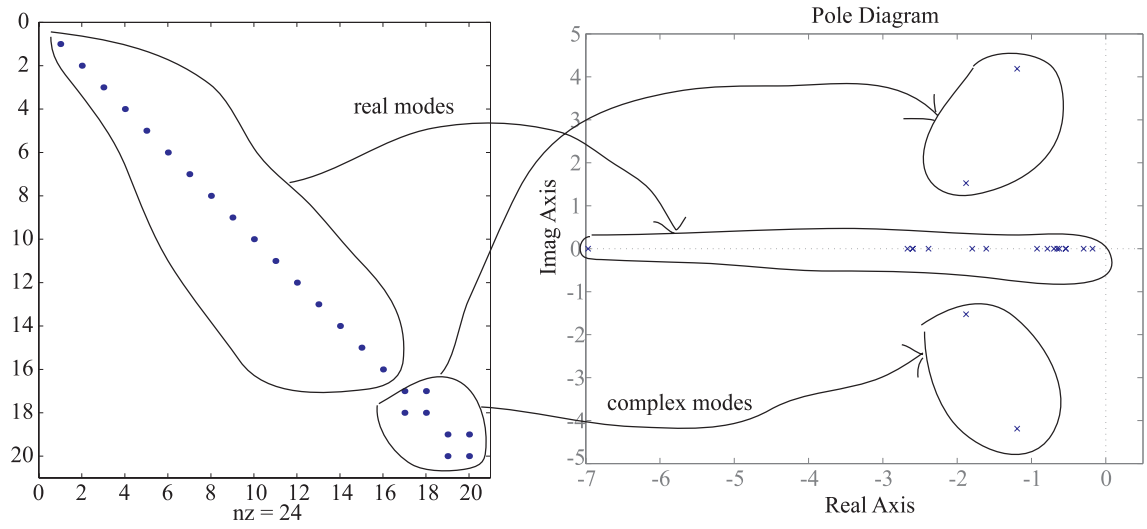


Figure 2-10: Sparsity plot and the corresponding pole diagram of a sorted diagonal A matrix

[7]For real modes, it is the magnitude of its eigenvalue that determines the speed of response, whether the mode is stable or unstable, which is determined by the sign of the eigenvalue.

[8]The generic form of a second order system is $\frac{k_n\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. ω_n is natural frequency, ζ is damping ratio and k_n is the modal gain.

transformed B and C matrices such that they are better numerically conditioned. On the other hand, if the system is defined such that the information like eigenvalues and mode shapes are known, it is possible to build the modal system without applying `ss2mod71.m`. One practical recommendation is that when the system is large (e.g. SIM), it is preferable to store (in sparse form if necessary) the diagonalized system for further use. The computation time of this diagonalization processing depends on the ability of the machine to compute eigenvalues and eigenvectors[9]. Since finding the eigenvalues of a matrix is equivalent to finding the roots of a polynomial, a problem to which closed form solution does not exist for order higher than 4 (by Abel), the algorithm to solve eigenvalue/eigenvector problems must be iterative and the convergence rate to the solution cannot be known a priori. Consequently, the comparison of computation cost between the diagonalization and direct simulation approach is not clear cut. Empirically the computation cost for diagonalization in FLOPS can be found by counting[10] the average FLOPS for a number of runs of an eigenvalue problem routine (e.g. `eig` in MATLAB). The result is given in Figure 2-11 and the approximate cubic law is

$$\text{FLOPS} = 27.8276 \times n_s^3, \tag{2.23}$$

where n_s is the number of state variables. As a comparison, the multiplications per state transition step for a dense A-matrix is $\mathrm{O}(n_s^2)$, but this number should be multiplied by the number of samples and the cost of discretization should be included for a fair comparison with the diagonalization approach. More details will follow in Subsection 2.4.5.

2.4.2 Subsystem Planner

The subroutines that implement this function are `seg_plan_real.m` and `seg_plan_com.m` for real mode and complex mode subsystems respectively. The objective of these subroutines is to create fictitious subsystems and assign each of them an "appropriate" sampling rate. Let's talk about the sampling rate issue first. Since computation effort is directly proportional to the number of samples (i.e. sampling rate when the time span of concern is fixed), it is a natural tendency to reduce the sampling rate as much as possible and the multiple sampling rates in `new_lsim.m` are multiple downsampling rates[11]. But as discussed in Section 2.3 when

[9] See http://www.ma.utexas.edu/documentation/lapack/node70.html for more details.

[10] The counting was done in MATLAB version 5.3.1, in which the routine `flops` is still available.

[11] Sampling rate is decreased by factors of 2,4,.... But in principle, these factors can be 3,5 or even any rational number [25].

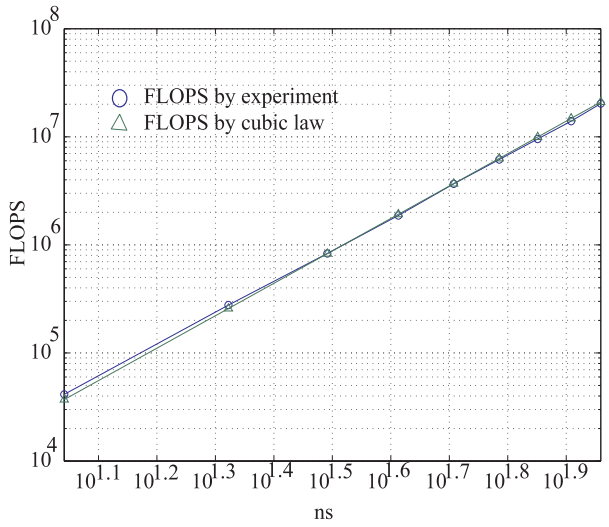


Figure 2-11: FLOPS for solving eigenvalues and eigenvectors. Circle: Actual FLOPS count. Triangle: FLOPS predicted by a cubic law.

the sampling rate is decreased, the risk of getting spectral overlap, or aliasing increases. The bound for the minimum "safe" sampling rate is given by the celebrated sampling theorem [25],

Theorem 1 (Nyquist Sampling Theorem) *Let $x_c(t)$ be a bandlimited signal with its spectrum $X_c(j\Omega) = 0 \quad \forall\ |\Omega| \geq \Omega_N$, where Ω_N is the bandwidth of the signal. Then $x_c(t)$ is uniquely determined by its samples $x[n] = x_c(nT),\ \forall\ n \in \mathbb{Z}$, if the sampling rate $\Omega_S = \frac{2\pi}{T} \geq 2\Omega_N$*[12].

Theorem 1 states the lower bound and does not provide the means to reconstruct the continuous-time signal from its samples. In fact, the reconstruction is accomplished through the following formula,

$$x_c(t) = \sum_{n=-\infty}^{\infty} x[n] \frac{\sin[\pi(t-nT)/T]}{\pi(t-nT)/T}. \tag{2.24}$$

An infinitely long sequence is involved in (2.24) and thus it is impossible in a real world application. An obvious alternative to the above ideal operation is linear interpolation. The performance of linear interpolation improves as sampling rate increases. This is clear if one understands that interpolation is low-pass filtering in the frequency domain and the frequency response as shown in Figure 2-12 is known. The frequency response is far from the ideal low-pass filter which corresponds to (2.24). However, as the sampling rate is increased (i.e. T in Figure 2-8 is reduced), the replications of the spectrum are more concentrated

[12]It should be noted that in some cases this might not be required. For example, for analytic signals (i.e. signals with one-sided spectrum), it is possible to reconstruct the signals from samples sampled at Ω_N.

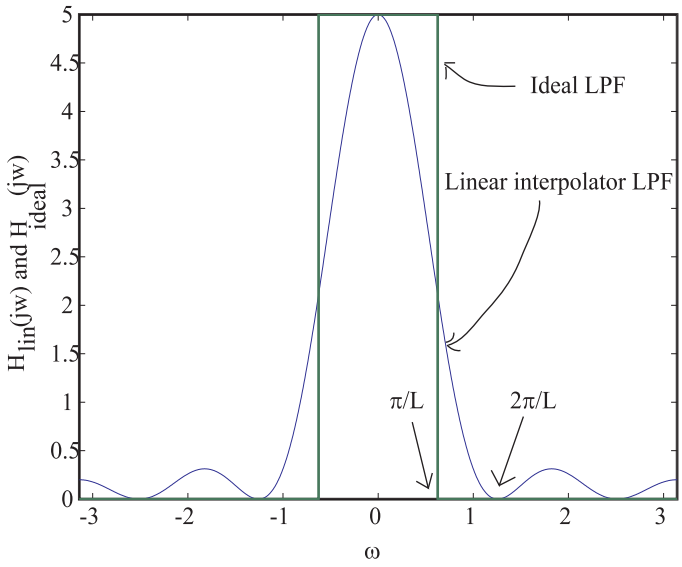


Figure 2-12: Frequency response of linear interpolator and ideal interpolator filters. Here L is the factor of interpolation and is numerically taken as 5.

at sampling frequencies (i.e. $0, \pm 2\pi, \pm 4\pi, \cdots$) and the approximation is more like the ideal low-pass filter. How well the approximation should be is not an easy question to answer and it depends on the problem at hand. *A rule of thumb is that the sampling rate should be four to ten times the system bandwidth when simulation is considered.* In fact, the default sampling rate chosen in `new_lsim` is four times the highest natural frequency of a subsystem but the factor can be changed by the user.

Now let's consider the issue of subsystem grouping. The assumption of this development is that the system is already block diagonal. *Before the discussion begins, it is vital to point out that the computation cost (time and FLOPS) for the simulation should increase linearly with the number of state variables, due to the block diagonal structure of the A-matrix.* In fact, the estimated simulation time has the simple form[13]

$$T_{cpu} = 2 \times (2 + m + p) \times n_s \times n \times \Delta t, \tag{2.25}$$

where m is the number of input channels, p is the number of output channels, n_s is the number of state variables, n is the number of samples to be processed and Δt is the average

[13]The operation count is really simple: By studying (2.8), it is clear that $Ax[n]$ requires approximately $2n_s$ multiplications per sample, $Bx[n]$ and $Cx[n]$ require $n_s \times m$ and $n_s \times p$ multiplications per sample respectively. The factor of 2 is added because there are approximately the same number of additions. Here feedthrough is neglected since the number of inputs and outputs is small compared the number of state variables.

time for computing one floating point operation. From the point of view of (2.25), it does not matter the size the subsystems are since the computation effort of simulating the whole system is the sum of the computation effort of simulating individual subsystems. However, the reality is, the estimator (2.25) is based on operation count and other issues such as memory traffic and initialization can bring the efficiency down, as will be explained in the following development.

A few things should be considered in realizing the subsystem grouping and they are illustrated through some numerical experiments that follow.

Efficiency regarding block size, number of input/output channels and samples. This is concerned with the problems that force the actual computation time to deviate from what is predicted in the ideal case. The first experiment is to simulate randomly generated modal systems (SISO) of various size in terms of state variables. The input to the system is also randomly generated and has the length of 1×10^5 samples. The computation time for each time chunk is given in Figure 2-13 (a)[14] and the corresponding efficiency ($\frac{T-T_d}{T_d} \times 100\%$, where T and T_d are the actual time and predicted time respectively. A positive value means the simulation takes more time than predicted.) relative to the linear law (2.25) is given in Table 2.2. Figure 2-13 (b) shows the equivalent time to simulate a

Table 2.2: Efficiency relative to linear law of simulating systems of various sizes. SISO case.

n_s	10	20	50	100	200	300	400	500
Eff (%)	682.5000	310.0000	142.0000	68.0000	54.3750	32.8333	34.7500	14.8500
n_s	600	700	800	1000	1200	1500	2000	
Eff (%)	15.8750	19.4286	11.8125	10.9500	15.8958	11.2000	17.9750	

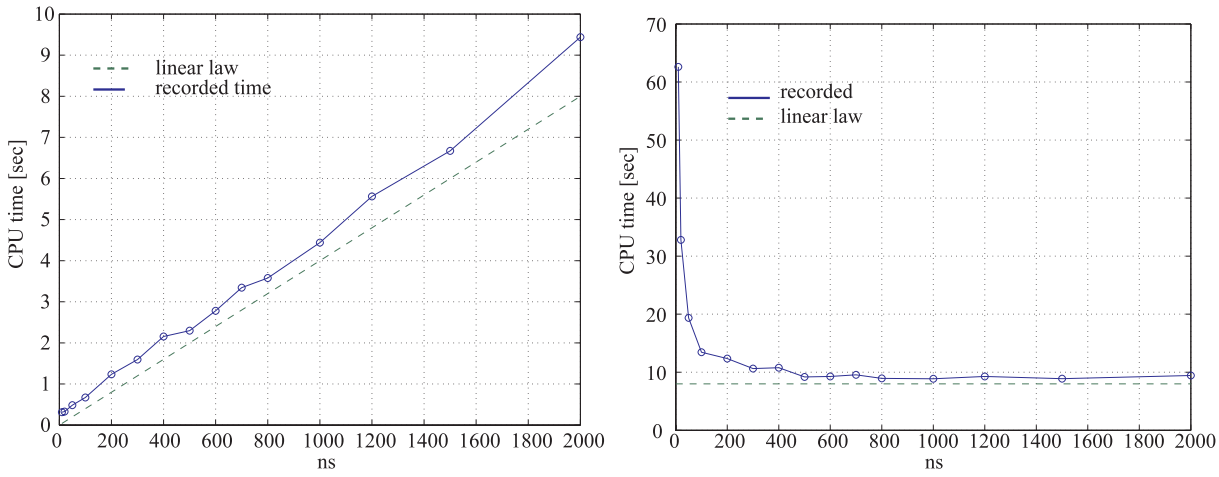
2000-state variable system (i.e. the recorded time in Figure 2-13 (a) is multiplied by $\frac{2000}{ss}$ where ss is the system size)[15].

Some conclusions can be drawn from the the results above.

1. The computation time in Figure 2-13 (a) is typically longer than the corresponding time estimated by the linear law. The reason for this phenomenon is that the linear

[14]The computation time per multiplication (Δt) needs to be fixed in order to actually give the dashed line. Δt is given as 5×10^{-9} s here after extensive check on the performance of the test machine (specification see Chapter 1).

[15]This equivalent time is not exact in that the matrix addition might include additional time for the case in which the system size is less than 2000 state variables. However, this is not supposed to be a big problem if the memory issue is well addressed, i.e. swapping is not necessary.



(a) (b)

Figure 2-13: SISO situation and 10^5 samples. (a) Computation time of each system and the time predicted by the linear law (2.25). (b) Equivalent computation time to simulate a 2000-state variable system.

prediction does not include overhead like initialization cost. Recall that Δt here is obtained empirically from tests of MIMO systems (rather the data shown in Figure 2-13). In case of MIMO systems, the dimension independent overhead accounts for less time in the computation and the efficiency will be closer to the theoretical value.

2. The efficiency of using very small block scheme to simulate a large-order system is poor. This is true partly because of the overhead for each simulation, which is independent of system size and partly because of the fact that the computation power of the computer is not fully utilized since parallel computation or pipelining[16] may be ruled out by the small dimensionality of the problem.

3. The linear computation time prediction is verified by the recorded computation time in Figure 2-13, provided that there is no other major burden that degrades the efficiency. One major concern is with the large subsystem scheme in that the large matrices involved can saturate available memory and slows down the computation significantly. For example, a 2000-state variable system is simulated with different block size scheme and the equivalent time is shown in Figure 2-14. Comparing Figure 2-14 with Figure

[16] A good example is given in [12] and it can be interpreted in the current context: If raw material is supplied to an assembly line continuously, then the assembly line can achieve its top speed. If the raw material supply is not steady, then the efficiency of the assembly line is not 100%. The idea applies to vector operation.

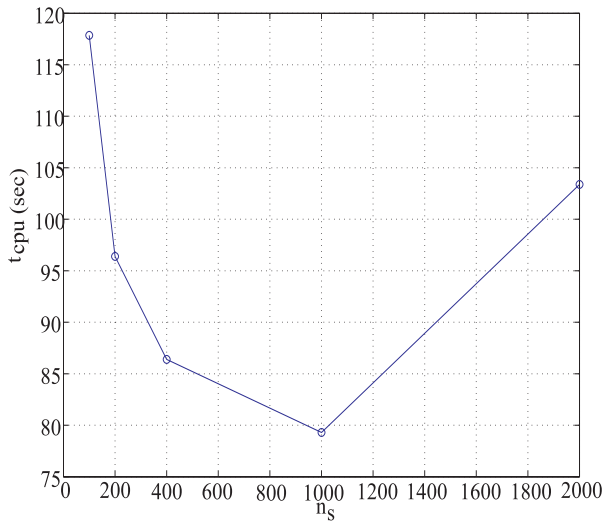


Figure 2-14: SISO situation and 8.6×10^5 samples. Equivalent computation time to simulate a system with 2000 state variables.

2-13 (b), it is clear that the 2000-state variable result is not robust in that it degrades significantly when the number of samples to be processed is higher. Therefore, a better scheme should have subsystem size smaller, preferably 800 to 1000 state variables.

4. A similar, but not identical case of large-order system analysis was studied in [6] in which the same decoupling approach was applied to facilitate the computation of the solution of the Lyapunov equation. There the optimal block size was found to be 20 to 100 state variables, much smaller than the "optimal" size here. In the case of solving large-order Lyapunov equations, the computation cost goes as $O(n_s^3)$ and it is reasonable that the block size is as small as possible, provided that the overhead does not dominate. However, in the case of the state transition of a block diagonal system with sparsity taken into account, the natural computation cost goes linearly with the dimension of the system and ideally there should not be an optimal (or equivalently, every point should be an optimal). The problems with small block and large block extremes are caused by something other than computation effort as suggested by the previous points.

The second experiment is similar to the first one. Here the test system is MIMO with 6 input channels and 6 output channels. The systems are randomly generated and the inputs are also randomly generated with fixed length 1×10^4 samples. The recorded computa-

tion time and the estimated time are in Figure 2-15 (a) and the corresponding equivalent computation time for simulating a 2000-state variable system is given in Figure 2-15 (b).

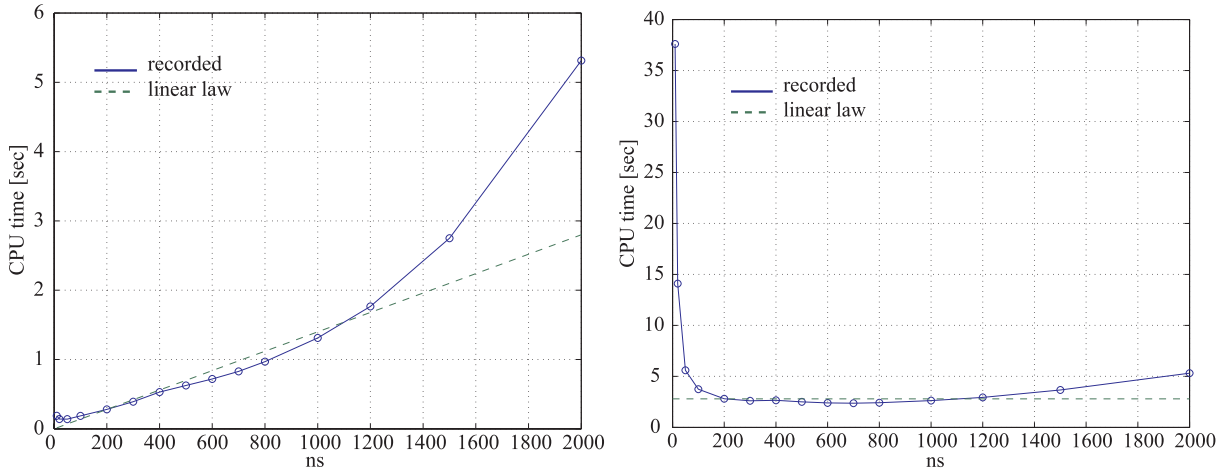


Figure 2-15: MIMO situation and 10^4 samples. (a) Computation time of each system and the time predicted by the linear law (2.25). (b) Equivalent computation time to simulate a 2000-state variable system.

The corresponding efficiency is listed in Table 2.3.

Table 2.3: Efficiency relative to linear law of simulating systems of various sizes. MIMO case and 10^4 samples.

n_s	10	20	50	100	200	300	400	500
Eff (%)	1242.9	403.5714	100.0000	33.5714	0.3571	-6.9048	-5.1786	-10.7143
n_s	600	700	800	1000	1200	1500	2000	
Eff (%)	-14.4048	-15.4082	-13.4821	-6.2857	5.1190	30.9524	89.7500	

Some conclusions can be drawn from the results above.

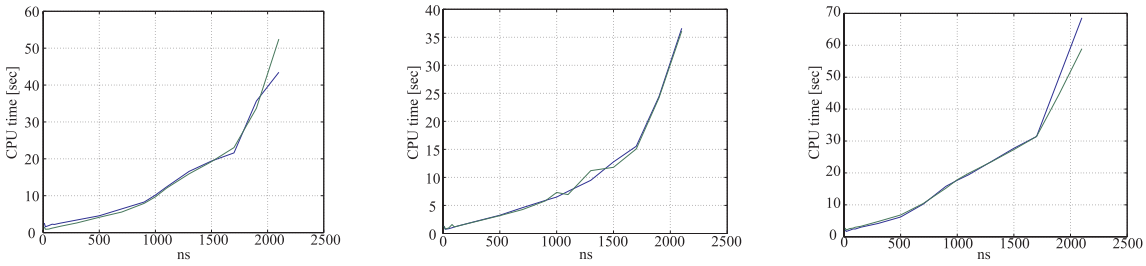
1. The small block scheme is still inefficient in the case of MIMO system simulation and is the result of the overhead.

2. The disadvantage of large block scheme shows up in that the computation time deviates obviously from the linear rule as in Figure 2-15 (a).

3. The "optimal" block size becomes smaller (600 to 800) with the increase in the number of input and output channels. In some cases, the recorded time gets even shorter

than the estimation by (2.25). Memory capability is again the main reason for this phenomenon. As can be predicted, the optimal block size will become smaller and smaller with even more increase in i/o dimensions. It should be pointed out, though, that the recorded time should be regarded as a sample of a random variable (RV) and the conclusions should not be viewed as absolute. Nevertheless, the variance of the RV is expected to be small (one does not expect a normal computer to perform the same job in strikingly different amount of time in different runs.) in the sense that the conclusions drawn from these experiments can be trusted with confidence.

4. The number of input and output channels affects the computation time roughly in the same way, as predicted by the linear law in (2.25). An auxiliary experiment has been conducted to verify this assertion. Here randomly generated systems are simulated with fixed length input (32768 samples) and the number of input and output channels varied. Three of the cases are shown in Figure 2-16. In these figures, the dimensions



(a) (b) (c)

Figure 2-16: Simulation time for MIMO case. (a) 3 and 5 i/o channels. (b) 7 and 5 i/o channels. (c) 17 and 11 i/o channels.

of input and output are exchanged and it is clear that the computation time increases are consistent as long as the total number of input and output channels remain the same. Here the computation time increase does not obey the linear law, especially for a large-order system, because of the same memory issue reflected in Figure 2-15.

Yet another experiment has been conducted to show the effect of the length of input samples. Here the system has 6 inputs and 6 outputs but the length of the input is 5×10^5 samples long. The efficiency relative to the linear time is listed in Table 2.4. Compared with Table 2.3 it is clear that a longer sequence generally deteriorates the situation.

The considerations above provide the motivation for the rules set by subroutine `seg_plan_x.m`.

1. A minimum block size is imposed to prevent the inefficiency due to small block scheme.

Table 2.4: Efficiency relative to linear law of simulating systems of various sizes. MIMO case and 5×10^5 samples.

n_s	10	20	50	100	200	300	400	500
Eff (%)	424.5714	233.7143	72.3143	22.0857	-2.5643	-11.2333	-11.5500	2.1886
n_s	600	700	800	1000	1200	1500	2000	
Eff (%)	-2.1929	-5.1653	-6.7536	-3.8171	8.4821	36.2352	100.1679	

2. A maximum block size is imposed to prevent the problems encountered by large block scheme.

3. After the subsystems grouping has been planned, the subsystems are further planned to suit the "optimal" block sizes listed in Table 2.5 (the discussion of the third row, namely, chunk length is deferred to Subsection 2.4.5.). The idea is that the compu-

Table 2.5: Optimal block size constrained by the number of i/o.

Input and output channels	SISO	≤ 4	≤ 8	≤ 12	≤ 16	≤ 24	> 24
Subsystem state variables	800	800	600	600	500	500	500
Chunk length (samples)	5×10^4	5×10^4	10^4	10^4	10^4	10^4	10^4

tation cost should be linear with the number of state variables as given in (2.25) and it is realized by appropriately choosing the size of the blocks and the length of each time chunk (see Subsection 2.4.5).

Sampling rate considerations. The above discussion is not the whole story since the simulations were carried out at a single sampling rate (i.e. the highest sampling rate). Multiple sampling rates are also a factor in determining the size of the subsystems since the smaller a subsystem is, the narrower its bandwidth is typically, so a better sampling rate can be assigned to the simulation. Another experiment was done to show this trend. See Table 2.6 and Figure 2-17. This time the systems simulated have different sizes in total, namely 50, 100, 200, 300 and 600. For each of these systems, simulations at different sampling rates are carried out and computation times are recorded. Each row here corresponds to the time required to simulate a system of particular size. It should be mentioned that the computation time here includes the time to interpolate (last step in Figure 2-4). The increasing time spent verifies the expectation that the more samples are processed, the

Table 2.6: Simulation time (in seconds) with different block size and sampling rate. The highest sampling rate is 4096Hz

Downsampling factor ($\frac{f_s}{f_d}$)	64	32	16	8	4	2	1
Actual sampling rate (Hz)	64	128	256	512	1024	2048	4096
Size in state variables							
100	1.3600	1.4530	1.6410	2.3440	4.5780	12.0620	38.0150
200	1.3900	1.5470	1.8590	2.6880	5.2960	13.8130	41.0930
400	1.5310	1.7500	2.2660	3.5620	7.0310	16.8750	48.1100
600	1.6880	2.0150	2.7500	4.4690	8.6250	20.3120	55.4840
1200	2.1560	2.7810	4.0620	6.9220	14.0470	30.9840	74.2660

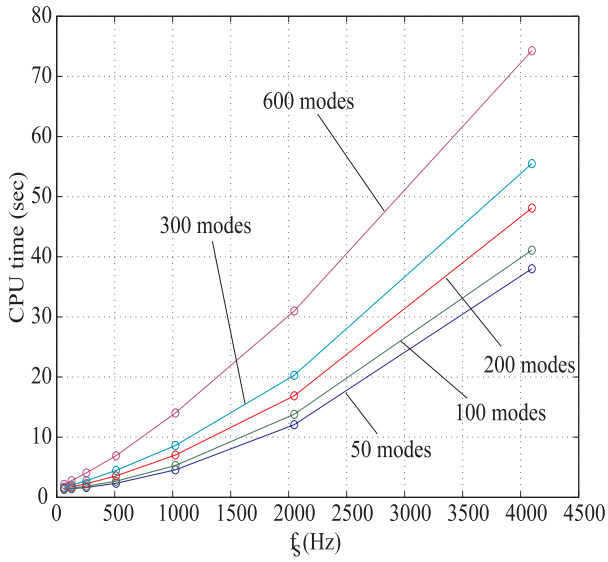


Figure 2-17: CPU time versus sampling rates with systems of different sizes.

more operations are required and so the computation time is longer. Also, this trend is independent of the size of the system simulated. If one pays close attention to the time for downsampling factors such as 64 and 32, one will discover that the time reduced is NOT proportional to the way the sampling rate is reduced. For example, halving the sampling rate does not result in halving the simulation time. A simple explanation is again the idea of pipelining that does not show an advantage in small-scale problems. The above study shows that too low of a sampling rate cannot achieve obvious gains in efficiency, but on the other hand can result in severe error. Therefore, it is not advantageous to create very small subsystems that are suitable for very low sampling rate. This is another motivation for having a lower bound on the subsystem size in the subroutine `seg_plan_x.m`.

It should be pointed out that downsampling is a compromise between efficiency and accuracy and it cannot be implemented unless the error is negligible compared with the prescribed level of tolerance. Also, the actual downsampling scheme implemented by `new_lsim.m` is different from the traditional scheme described above so that the accuracy is much better at the expense of less gain in efficiency (see Subsection 2.4.3).

With all the above considerations, it is now possible to explain the actual subroutine for the subsystem planning. Figure 2-18 is the flow chart of this subroutine. In the flow

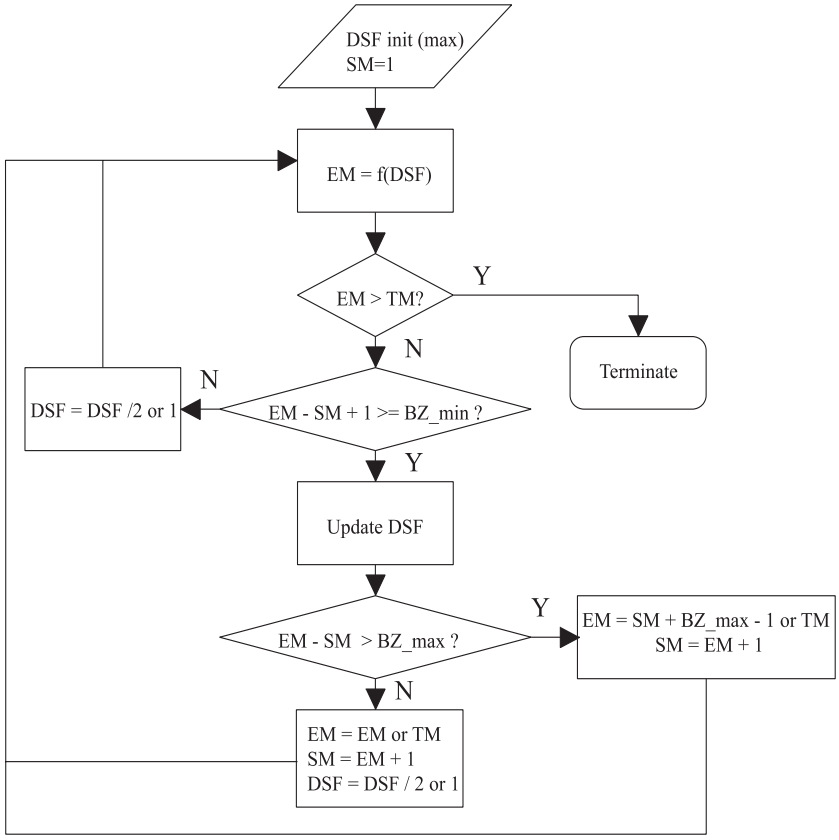


Figure 2-18: Flow chart of `segment_plan_x.m`. DSF: downsampling factor. SM: start mode. EM: end mode. TM: total modes of the original system. BZ: block size.

chart, DSF stands for downsampling factor. SM_i and EM_i are the starting and ending modes of a particular subsystem. TM is the total number of modes of the original system. This primarily serves as a condition to terminate the loop in the subroutine. BZ_i is the block size of a subsystem. The input to this subroutine is the system with its modes sorted and the outputs are a matrix with the information of the decisions for each subsystems (i.e. the starting mode, ending mode and its sampling rate). The principle of this logic is

as follows. There should be lower and upper bounds of a subsystem. Also, downsampling should be employed to simulate as many modes as allowed by the error tolerance. Another important characteristic is that the computation in this subroutine should be cheap. The subroutine starts with the first mode of the system and maximum allowable downsampling factor given by the user (the variable "dsf–init" and its default value is 4). Then it calculates the ending mode of the first subsystem with the criterion that the sampling rate should be an integer (the variable is "resolution") times the natural frequency of the ending mode. Also, the subroutine tries to estimate the error in the output due to ignored high frequency components by sampling the frequency response at different frequencies and the spectrum of the external input $u(t)$[17]. In `new_lsim`, the variable "freq–error" is the desired percentage of this error and its default value is 1%. Of course, the calculated ending mode should be checked to see if all the modes have been considered (i.e. the termination of the subroutine). If not, the lower bound constraint is checked to verify whether it is violated or not. If so, then the subroutine chooses another downsampling rate to start again. If not, the subroutine records the downsampling factor to be used in simulations that follow. After that, the ending mode is checked against the upper bound of a subsystem and the starting mode and a sampling factor for the next iteration is decided according to the two possible outcomes. If the upper bound is violated, then it means the current sampling can be used for the next iteration and the ending mode is decided according to the upper bound rather than that calculated in the beginning of the iteration. If the upper bound is not violated, then the calculated ending mode is recorded and the sampling rate of the next iteration is updated. With the sampling rate and starting mode, the next iteration begins and the loop continues until the last mode of the system is considered. A few examples are shown here to illustrate the ideas just discussed. A 1086-mode (or 2172 state variables) SIM model v2.2 is analyzed. If the maximum downsampling factor is set to 4, the desired error is 1% and the sampling rate (the original sampling rate is 4096 Hz) should be 4 times the highest natural frequency of each subsystem, then the result is as follows.

```
>> segment = seg_plan_com(sys_c.a,sys_c.b,sys_c.c,sys_c.d,u,1/4096,1/100,6,4)
segment =
           1         601        1026            % first mode
         600        1025        1086            % last mode
           4           4           1            % DSF
```

[17]This is really a dilemma: the approximation error can be known precisely only after the simulation, when the prediction is no longer meaningful.

```
ss1          ss2          ss3              % subsystem
```

In the command above, sys–c.x is the system matrix (ABCD) and u is a random input. The result is interpreted as follows. The first row is the starting mode of each subsystem, the second row is the ending mode of each subsystem and the last row is the corresponding downsampling factor. Figure 2-19 is the graphical representation of this grouping. Figures 2-20 (a) and 2-20 (b) show the magnitude Bode diagram of the subsystems and the fact that the error due to ignored high frequency components will be less than 1% if the downsampling scheme is adopted. Another example is a 1000-mode system with linearly increasing natural

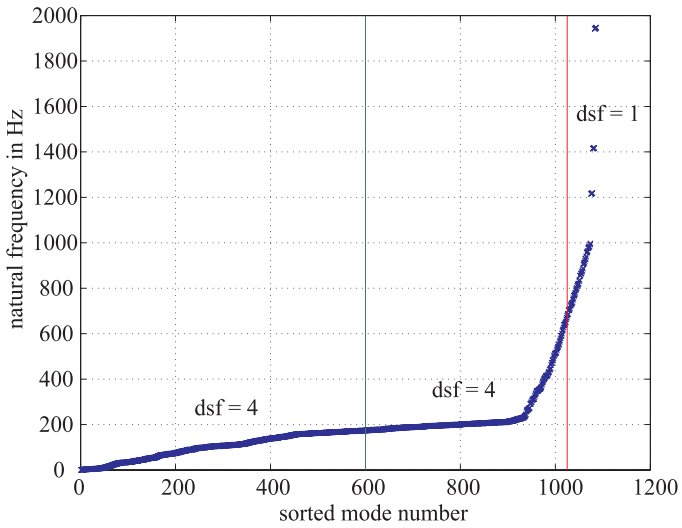


Figure 2-19: Subsystem planning of SIM v2.2

frequencies and the result is as follows. Here u is a random signal.

```
>> segment = seg_plan_com(a,b,c,d,u,1/4096,1/100,6,4)
segment =
           1         601
         600        1000
           4           2
```

If the error tolerance is reduced from 1% to 0.5%, then the second subsystem cannot be simulated at half sampling rate (i.e. $\frac{4096}{2} = 2048$ Hz).

```
>> segment = seg_plan_com(a,b,c,d,u,1/4096,0.5/100,6,4)
segment =
           1         601
         600        1000
           4           1
```

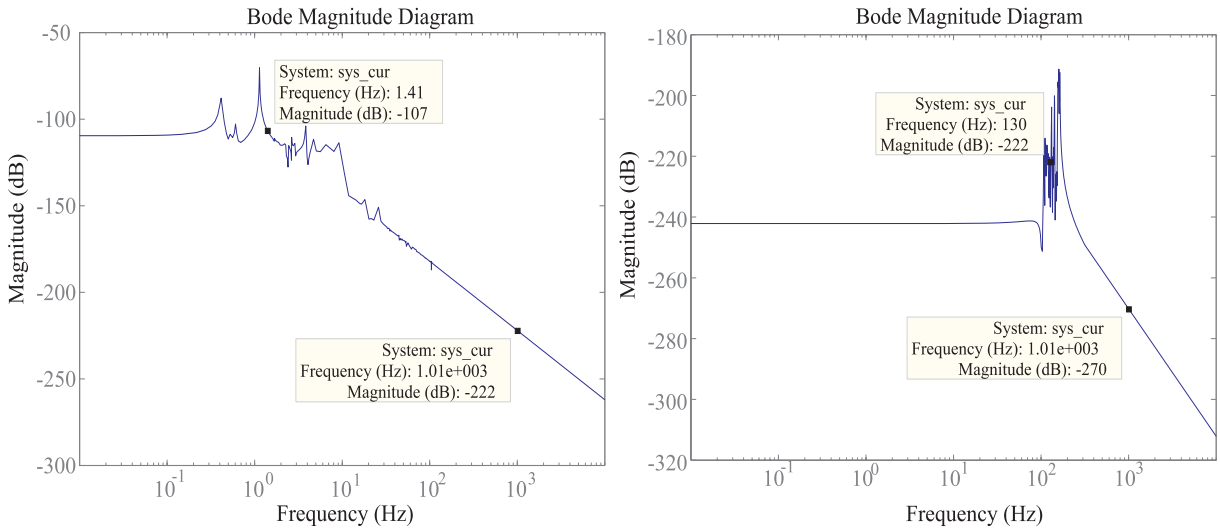


Figure 2-20: Magnitude Bode of the subsystems with its characteristic magnitude response. (a) First subsystem. (b) Second subsystem. Lower frequency point denotes where the dominant response will be and higher frequency point denotes the cutoff frequency of the subsystem.

If the signal is changed, say, to a very low frequency one like $\cos(0.2\pi t)$, then the result is different.

```
>> segment = seg_plan_com(a,b,c,d,u1,1/4096,1/100,6,4)
segment =
           1         601
         600        1000
           4           4
```

Figure 2-21 (a) shows the grouping graphically. Obviously the restriction on the sampling rate is due to consideration of ignored frequency components. Another example is given here to show what natural frequencies of the modes can do to restrict the choice of downsampling factors. The grouping is shown in Figure 2-21 (b) and it is obvious that in this case, the natural frequency prevents downsampling treatments for the second and third subsystems.

2.4.3 Downsampling

In this subsection the issue concerning downsampling (subroutine `build_multirate_sys.m`) is discussed. The notion of downsampling is as follows: If the input to a downsampler is $x[n]$, then the output $y[n] = x[nM]$, where M is the downsampling factor. Figure 2-22 shows the time domain representations of the signal and its downsampled version. Before

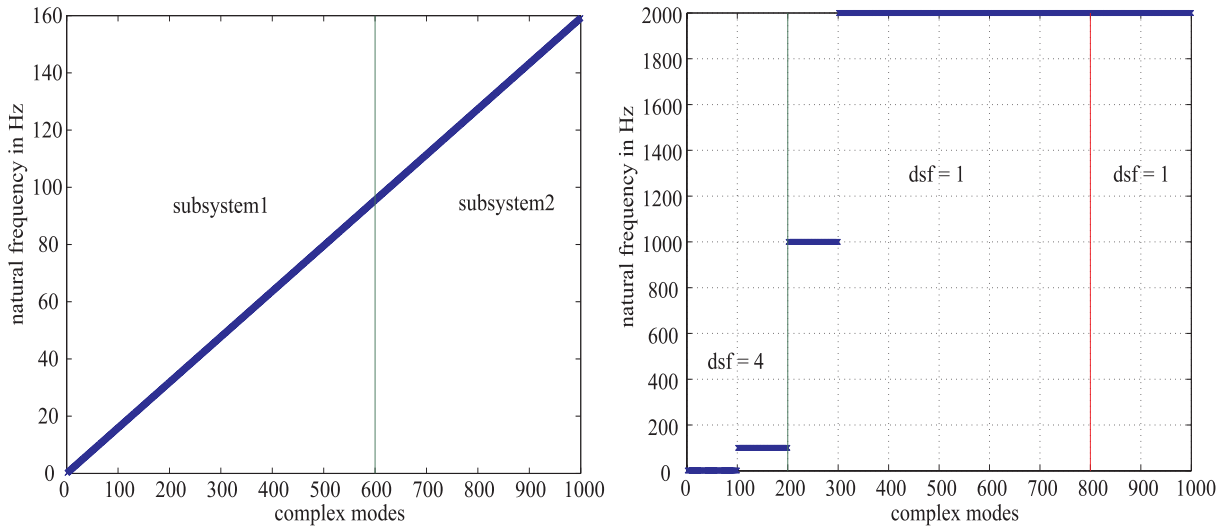


Figure 2-21: (a) Subsystem planning of a system with linearly increasing natural frequencies. (b) Subsystem planning of a system with plateaus of natural frequencies.

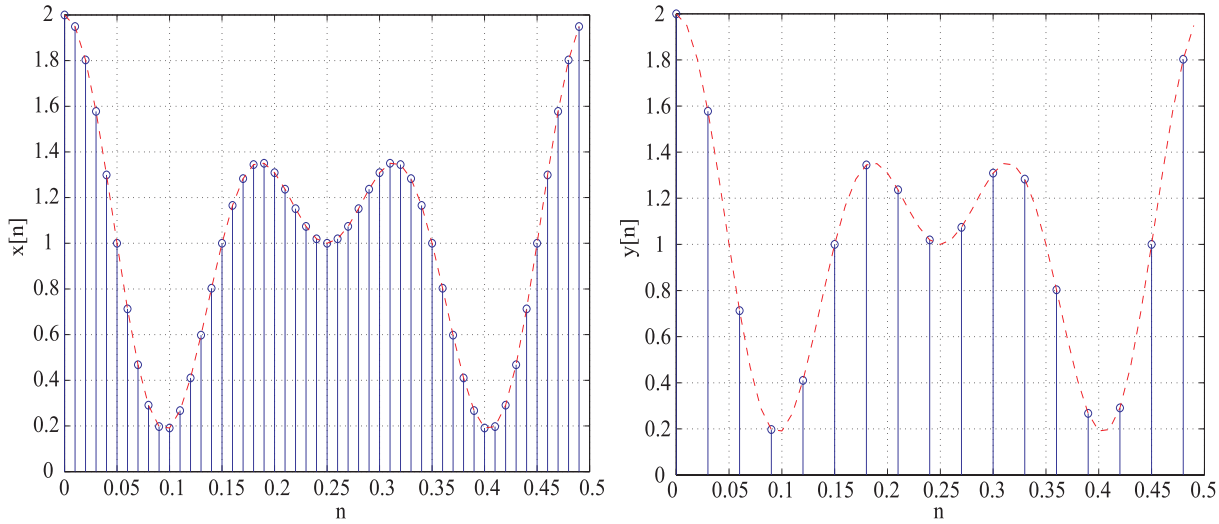
the elaboration on the details of downsampling, a few questions should be resolved,

- Why is downsampling needed anyway? Because it is desirable (especially in the case of large-order systems) to reduce the computation effort by reducing the sampling rate of the simulation (i.e. that of the signals[18] involved).

- Why is it allowable to downsample? Because of the low-pass nature of physical systems and signals, low frequency components of the response of the systems usually dominate. According to the idea of bandwidth of signals and the frequency domain representation discussed in Section 2.3, a relatively lower sampling rate is enough to represent the signal.

- To what extent can signals be downsampled? It depends on the nature of the signal and the allowable error tolerance. For example, if the signal is white noise (i.e. a stochastic process whose PSD is uniform over all range of frequencies concerned), then it does not make sense to downsample since the original signal cannot be deduced from its sampled version. On the other hand, if the signal is slow varying such as $\cos(2\pi t)$, then it is natural that nearby samples are closely related so the original signal can be recovered to a satisfactory level of accuracy. The estimation for the error is more

[18]The signal related to a system is its impulse response.



(a) (b)

Figure 2-22: (a) Waveform of the original signal and its envelope. (b) Waveform of the downsampled signal and its envelop. Downsampling factor here is 3.

difficult to answer. Certain criteria should be used to make the decision. For example, the bound on the ignored frequency components relative to that of the kept frequency components would be an appropriate choice. In addition to that, the energy (or power) ratio can also be employed to determine the significance of the ignored signals.

Now let's discuss the way to implement the downsampling. There are two ways to analyze the input and output relationship of a dynamical system, namely time domain method and frequency domain method. These two perspectives can lead to two very different approaches to downsampling. Let's talk about the frequency domain approach first, since it is more straightforward. Let u(t) be the input to a system, h(t) be the impulse response of the system and y(t) be the output. The corresponding Fourier transforms of these signals are $U(j\omega)$, $H(j\omega)$ and $Y(j\omega)$. Then the transforms are related by

$$Y(j\omega) = H(j\omega)U(j\omega). \tag{2.26}$$

$H(j\omega)$ in (2.26) can be viewed as a filter that suppresses certain frequency components of $U(j\omega)$. If $H(j\omega)$ is low-pass, then the high frequency components of $U(j\omega)$ contribute only very little to $Y(j\omega)$ and can be ignored. The situation is depicted in Figure 2-23. Hence, all the signals involved are bandlimited and a low sampling rate is enough to represent them accurately. To sum up, the whole idea above leads to the following implementation.

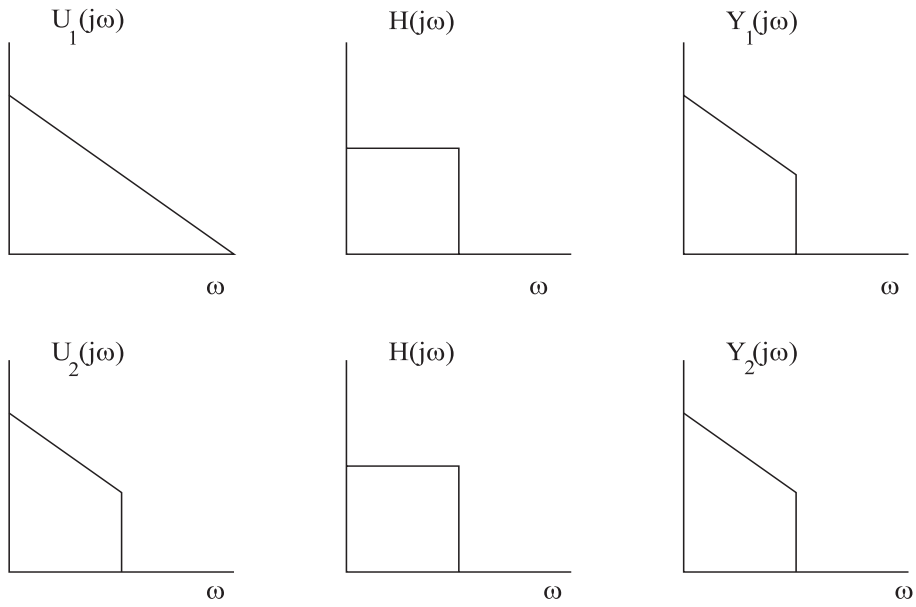


Figure 2-23: If $H(j\omega)$ is ideally bandlimited, then $Y_1(j\omega)$ and $Y_2(j\omega)$ are the same provided that the low frequency part of $U_1(j\omega)$ and $U_2(j\omega)$ are same.

First the input signal is decimated (i.e. low-pass filtered and then downsampled[19]), then the simulation is carried out at a low sampling rate. If this implementation is employed, then an efficiency gain by the factor of the downsampling ratio can be achieved, provided that the necessary filtering is not expensive and the original input sequence has a significant number of samples (see Subsection 2.4.2). The approximate computational cost in CPU time will be

$$T_{cpu} = 2 \times \frac{(2 + m + p) \times n_s \times n}{\text{DSF}} \times \Delta t, \tag{2.27}$$

where m is the number of input channels, p is the number of output channels, n_s is the number of state variables, n is number of samples, DSF is the downsampling factor and Δt is the average time to compute one FLOP. However, this method suffers from some problems with accuracy and they are listed briefly as follows:

1. There are no ideally bandlimited signals and systems. Hence the spectra of the downsampled signals must overlap with each other.

2. Ideal low-pass filtering is impossible in real world applications. This introduces further imperfection to the accuracy of the filtered input signal.

[19]The low-pass filtering is usually necessary, otherwise aliasing can corrupt the result significantly.

3. Discretization error arises whenever a downsampled simulation is employed. For example, if the discretization scheme is zero order hold and the downsampling factor is 2, then the downsampled simulation implicitly assumes that the input signal is constant over every two time steps. This assumption is not usually true and this will induce additional error relative to the state transition scheme at full sampling rate. It should be noted that even if the input signal is low-pass filtered, it does not mean that it is constant over the larger time step due to downsampling.

Therefore, the direct downsampling approach resulting from frequency domain analysis will not be employed by `new_lsim.m`. Nevertheless, the idea of this downsampling approach is useful in a quick check of the relative significance of the high frequency components of the output signal and it is used by the subroutine `seg_plan_x.m` to estimate the bound for downsampling factor.

Now let's study the second approach to downsampling due to time domain analysis. Recall the state transition equation (or the state equation for the discretized system),

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n]. \end{aligned} \tag{2.28}$$

The first equation in (2.28) is certainly satisfied at the $n+1$ instant,

$$x[n+2] = A_d x[n+1] + B_d u[n+1]. \tag{2.29}$$

If (2.28) is substituted into in (2.29), then the following results

$$\begin{aligned} x[n+2] &= A_d \left\{ A_d x[n] + B_d u[n] \right\} + B_d u[n+1] \\ &= A_d^2 x[n] + A_d B_d u[n] + B_d u[n+1]. \end{aligned} \tag{2.30}$$

(2.30) can be generalized for N steps,

$$\begin{aligned} x[n+N] &= A_d^N x[n] + A_d^{N-1} B_d u[n] + \cdots + B_d u[n+N-1] \\ &= A_d^N x[n] + \left[\begin{array}{ccc} A_d^{N-1} B_d & \cdots & B_d \end{array} \right] \left[\begin{array}{ccc} u[n] & \cdots & u[n+N-1] \end{array} \right]^T. \end{aligned} \tag{2.31}$$

If the following new matrices are defined

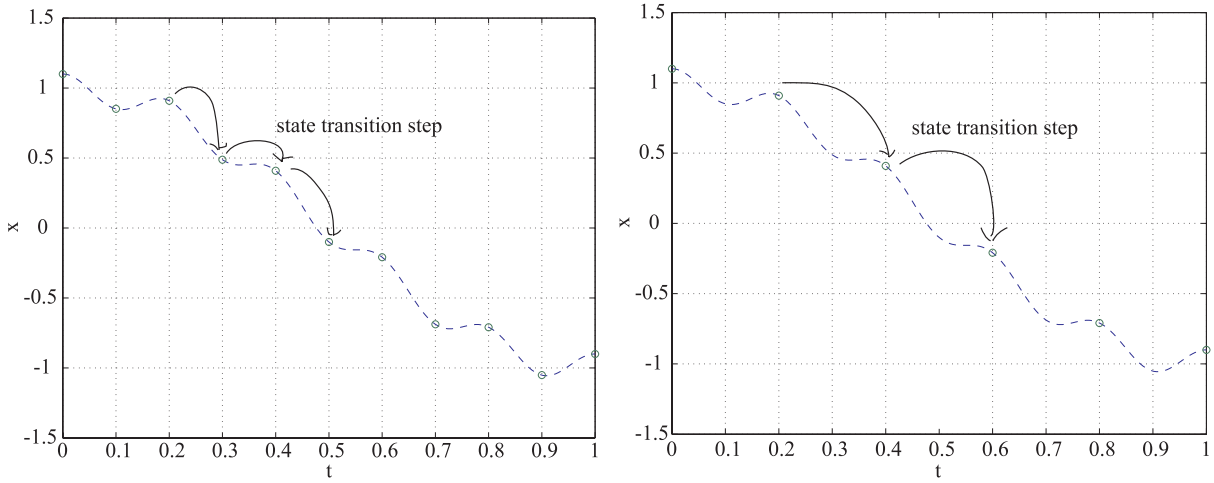
$$\begin{aligned} \underline{u}[n] &= \left[\begin{array}{ccc} u[n] & \cdots & u[n+N-1] \end{array} \right]^T \\ \underline{A_d} &= A_d^N \end{aligned}$$

$$\begin{aligned} \underline{B_d} &= \left[\begin{array}{ccc} A_d^{N-1}B_d & \cdots & B_d \end{array} \right] \\ \underline{C} &= C \\ \underline{D} &= \left[\begin{array}{cccc} D & 0 & \cdots & 0 \end{array} \right], \end{aligned}$$

then the N-step propagation version of (2.28) is

$$\begin{aligned} x[n+N] &= \underline{A_d}x[n] + \underline{B_d u}[n] \\ y[n] &= \underline{C}x[n] + \underline{D u}[n]. \end{aligned} \tag{2.32}$$

(2.32) is graphically shown in Figure 2-24. By employing a long time step state transition,



(a) (b)

Figure 2-24: Schematic of state transition. (a) Short time step (high sampling rate). (b) Long time step (low sampling rate).

the calculation of the unwanted intermediate state and output can be avoided. The generalization of this downsampling procedure is lifting [33] and will be discussed further in Chapter 3. The main difference between the downsampling schemes due to time domain and frequency domain analysis is that in the frequency domain case, the input is downsampled, while in the time domain case, the output is downsampled. Using the second approach, the problems with the imperfection of low-pass filtering and discretization error can be avoided, while the error due to downsampling a band-unlimited signal is inevitable in both cases. Nevertheless, the disadvantage of the second method compared with the first one is obvious in that the downsampling is not as efficient, as the following estimate for the computation cost reveals

$$T_{cpu} = 2 \times \frac{(2 + DSF \times m + p) \times n_s \times n}{DSF} \times \Delta t. \tag{2.33}$$

Evidently the saving of computational effort depends upon the ratio between the number of input channels and the number of output channels. For example, if the input-downsampling approach is adopted and the downsampling factor is 4, then theoretically the computational effort will be reduced to $\frac{1}{4}$ of the original amount. For the output-downsampling method, computational effort reduces only to $\frac{7}{8}$ of the original amount for a SISO system and $\frac{4}{7}$ for a MIMO system with 6 inputs and 6 outputs, according to (2.33). One last point about the second downsampling scheme is that additional computation time can be induced during the process of defining $\underline{u}[n]$ from $u[n]$ in that the order[20] of the entries of these two matrices are not the same, which implies that additional memory swapping is necessary. The issue becomes more important as the matrices get large. To sum up, there are pros and cons with the two downsampling approaches (downsampling input and downsampling output), the scheme to downsample the input is more aggressive but the error is serious, on the other hand, the scheme to downsample the output is more accurate but is less efficient, especially for SISO case. For this version of `new_lsim.m`, the more conservative method is adopted.

2.4.4 Discretization

In this subsection, the issue of discretization will be discussed. The m-file that implements the operation is `build_multirate_sys.m`. Since the A-matrix of a subsystem is block diagonal, it is possible to divide the work of discretization into several parts so that the Padé approximation is cheap to compute. In fact, the subroutine implements a "for" loop over all the modes of a subsystem and uses MATLAB command `c2d.m` to discretize them. The reason why `c2d.m` is again chosen instead of (2.22) in Section 2.4 is that `c2d.m` allows the subroutine to compute more discretization schemes (e.g. first order hold), although in this version of `new_lsim.m` the discretization scheme is only zero order hold (ZOH)[21].

2.4.5 Simulation, Interpolation and Superposition

Simulation

After all preparation steps discussed in previous subsections, the discrete-time state equations and output equations (2.28) are ready and now it is the job of a simulator to compute

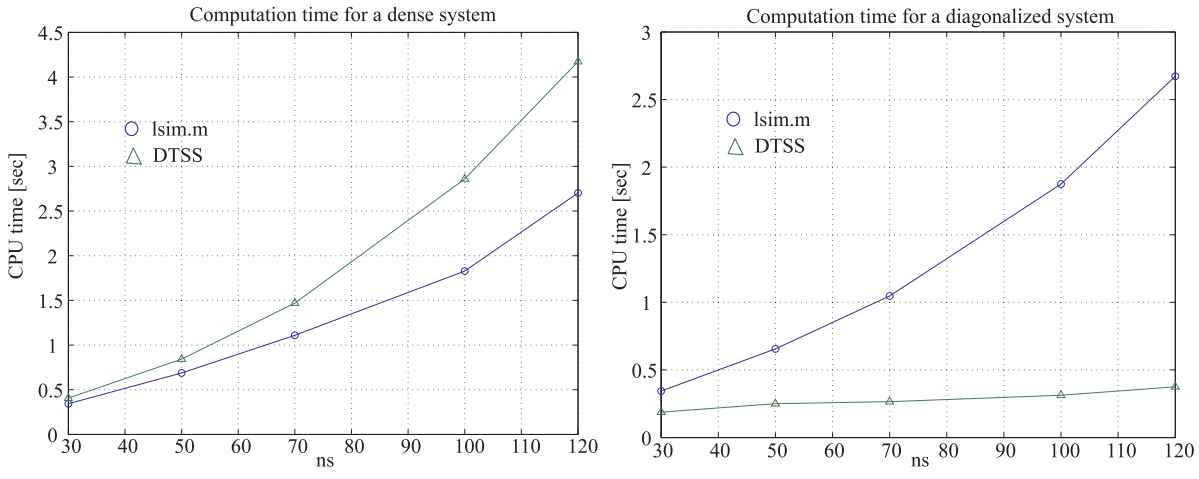
[20] For example, for a 2D array (matrix) $[a_{ij}]$, only one of these two cases can be true: a_{ij} and a_{ij+1} have neighboring addresses or a_{ij} and a_{i+1j} have neighboring addresses provided that the indexes make sense.

[21] FOH is much more complicated in that not only A and B matrices are changed, the C-matrix, D-matrix and the initial conditions are also changed.

the response. There are a few considerations regarding the simulator: accuracy, efficiency and robustness.

The accuracy of a typical commercial computation program like FORTRAN and MATLAB should not be a problem and there are few things a user can do to alter this fact.

The efficiency, on the machine level, is fixed. For example, the speed for multiplication of floating point numbers is determined roughly by the speed of the processor. At algorithm level, however, the simulation efficiency can differ due to different computation strategies. For instance, the computational cost for simulating a system with block diagonal A-matrix is linear as given by (2.25), only if the simulator can take advantage of the inherent sparsity[22]. In MATLAB, for example, `lsim.m` does not recognize diagonal structure of the A-matrix, so the computation cost goes with $O(n_s^2)$. On the other hand, the discrete-time state-space (DTSS) block in SIMULINK converts the matrices into sparse form and the special structure of these matrices is exploited, thus resulting in linear computation cost for a diagonal system. An example of two solvers that exploit the structure of the A-matrix is given in Figure 2-25. Here in Figure 2-25 (a) the system has an ordinary dense A-matrix, while in Figure 2-25



(a) (b)

Figure 2-25: (a) Computation time for simulating a system whose A-matrix is dense. (b) Computation time for simulating the diagonalized system. The simulation time for `lsim.m` remains nearly unchanged but that of `DTSS` changes a lot.

(b) the system was obtained by applying a similarity transform (diagonalization) to the one

[22]The situation happens in other fields too. For example, the source coding theorem states the maximum allowable transmission rate from a given source, but how to achieve the bound is the job of coding algorithms, a subject that requires as much ingenuity as the discovery of the theorem itself.

in subplot (a). Note the difference in computation time. Note also that in Figure 2-25 (b) the computation time curve for DTSS is close to linear, as predicted.

Another issue concerning the solvers is robustness, which means reliability in the current context. In the specific case of large-order system simulation, a solver may fail if the scheme for storing intermediate variables does not take into account the dimensionality of the system. For example, `lsim.m` in MATLAB computes and stores the state in a matrix and then computes the output by simple matrix-matrix multiplication. This strategy causes a serious problem (the solver fails because it fails to find a large piece of contiguous memory[23]) when the system order is high, see Chapter 1 for more detail. On the other hand, DTSS stores the variables in a buffer and options are provided such that unimportant information such as intermediate states can be thrown away to achieve economical use of memory.

Because of the particular problem (large-order and block diagonal) and the simulator characteristics (efficiency and robustness), DTSS is preferred over `lsim.m`. Nevertheless, the choice of solvers should not be restricted only to those provided by MATLAB. Any solver that has the desired properties for the problems at hand is appropriate. The reason why only `lsim.m` and DTSS are focused on is that the routine `new_lsim.m` is, at least for this version, based on MATLAB.

Due to the way SIMULINK stores its output variables[24], the simulation time can be further reduced if the simulation is divided into several parts. To be precise, the effect of time division should be parsed as follows: the additional computational burden due to the solver can be avoided by time division so that the computation time can remain linear with the number of state variables if the A-matrix of the system is block diagonal. An example of this time truncation scheme is as follows: if the time span is 100 seconds, then two simulations of 50 seconds will generally have better efficiency. The main reason for this phenomenon is that output variables are stored in a limited size buffer before written to workspace when the simulation stops, so division of the time history reduces the possibility of running into memory saturation. The trick of splitting the time span is referred to as the "split time technique" by the author. Now let's contrast the difference between the performance of the split time technique and direct simulation scheme in Tables 2.7 and 2.8.

[23] See http://www.mathworks.com/support/tech-notes/1100/1106.shtml for a description of the memory issues for MATLAB.

[24] See http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/ug/toworkspace.shtml for more information.

Table 2.8 is the same as Table 2.6 in Subsection 2.4.2. The way the experiment is done is exactly the same as that in Subsection 2.4.2. The only difference here is that Table 2.7 is obtained by using the split time technique. As shown from the tables, the split time

Table 2.7: Simulation time of systems of different sizes at different sampling rate using split time technique

Downsampling factor ($\frac{f_s}{f_d}$)	64	32	16	8	4	2	1
Size in state variables							
100	1.4060	1.5000	1.7030	2.1250	2.8120	4.2340	5.2810
200	1.5160	1.6870	2.0000	2.5320	3.6410	5.7810	8.2660
400	1.9530	2.1870	2.7500	3.6720	5.4840	9.2030	14.8910
600	2.5780	2.9690	3.7500	5.2660	8.3290	13.5940	21.7190
1200	5.5780	6.1250	7.3750	9.8750	14.6720	24.3280	42.4370

Table 2.8: Simulation time of systems of different sizes at different sampling rate without using split time technique

Downsampling factor ($\frac{f_s}{f_d}$)	64	32	16	8	4	2	1
Size in state variables							
100	1.3600	1.4530	1.6410	2.3440	4.5780	12.0620	38.0150
200	1.3900	1.5470	1.8590	2.6880	5.2960	13.8130	41.0930
400	1.5310	1.7500	2.2660	3.5620	7.0310	16.8750	48.1100
600	1.6880	2.0150	2.7500	4.4690	8.6250	20.3120	55.4840
1200	2.1560	2.7810	4.0620	6.9220	14.0470	30.9840	74.2660

technique is usually faster. Figure 2-26 is a graphical illustration of the advantage of the split time technique. Note also that the time achieved by the splitting time technique is linear with the number of samples, again as predicted by (2.25). The data is taken from row 4 of table 2.7 and 2.8. That is, the system has 600 state variables and the downsampling factors are 64, 32, 16, 8, 4, 2 and 1. The subroutine `st_sim.m` in `new_lsim.m` automatically computes the division provided that the time sequence is long enough. The rule for the time truncation is determined empirically and is given in Table 2.5 (the third row).

Interpolation

With the responses of the subsystems computed, they can be superposed to form the total response. However, before this could be done, it is necessary to interpolate the responses if they are obtained from a downsampled simulation, thus rendering the same number of

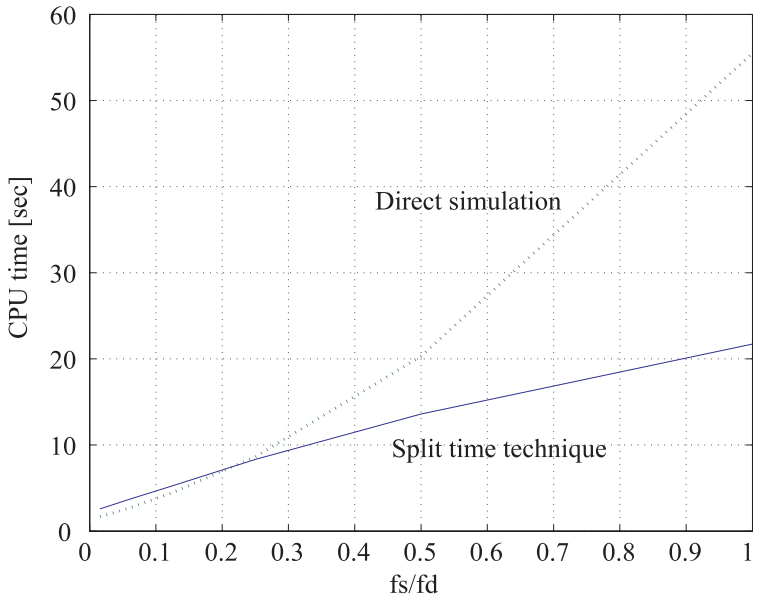


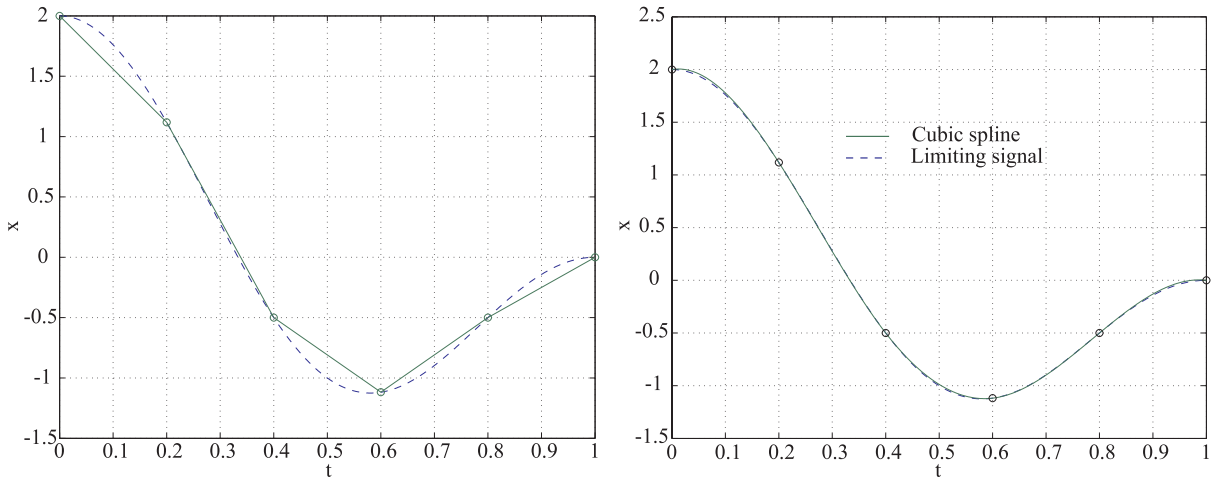
Figure 2-26: Simulation time at different sampling rates. Solid line: Split time technique. Dash line: Direct simulation. f_s is the highest sampling rate and f_d is the downsampled rate.

samples for the responses of each subsystem. There are a variety of interpolation methods available (see [5]). Linear interpolation, cubic spline, cubic Hermite are some popular choices (MATLAB command `interp1.m` for 1-D interpolation). For example, Figure 2-27 (a) shows the results of the linear interpolation. The choice of the interpolation method is again a tradeoff between computational effort and accuracy. Let's first address the issue of accuracy.

Also shown in Figure 2-27 (a) is the limiting signal as the interpolation step is getting finer and finer, as the following inequality shows,

$$\|x - L(x)\| \leq \frac{1}{8} |\Delta t|^2 \left\|x''\right\|, \tag{2.34}$$

where x is the limiting signal, $L(\cdot)$ denotes the linear interpolation operator, Δt denotes the interpolation step and x'' denotes the second order derivative of x with respect to t. While Δt is fixed (downsampling decision is made before the simulation, see Figure 2-4), the accuracy (the distance between the interpolated signal and the limiting signal) can be very different due to different choices of interpolation method. For example, choosing cubic spline can improve the accuracy significantly, as shown in Figure 2-27 (b), where the distance between the interpolated signal and its limit is not obvious. The following bound



(a) (b)

Figure 2-27: (a) An example of linear interpolation and its limiting signal. (b) An example of cubic spline and its limiting signal.

confirms the assertion,

$$\|x - S(x)\| \leq \frac{5}{384} |\Delta t|^4 \left\| x^{(4)} \right\|, \tag{2.35}$$

where x is the limiting signal, $S(\cdot)$ is the cubic spline operator, Δt is the interpolation step and $x^{(4)}$ is the fourth order derivative of x with respect to t. The story of accuracy is not finished yet, since a fundamental question concerning the application of downsampling in the context of the thesis remains to be answered. If a discrete-time signal $x[n]$ is downsampled and a reconstruction signal $\hat{x}[n]$ is obtained by interpolating the downsampled version of $x[n]$ (i.e. $x[Mn]$ where M is the downsampling factor), then $\hat{x}[n]$ converges to a limiting signal as $\Delta t \to 0$ or the interpolation method gets more sophisticated (e.g. cubic spline instead of linear interpolation), does $\hat{x}[n]$ converge to $x[n]$? In order to answer this question, it is necessary to carry out a frequency domain analysis. Let's study the block diagram in Figure 2-28, which is the graphical equivalence of the question just raised. The discrete-time Fourier

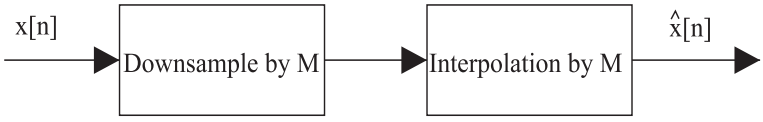


Figure 2-28: Block diagram of downsampling followed by interpolation.

transforms (DTFT) of the input and output of the downsampler are graphically shown in Figure 2-29. That is, the DTFT of the output of the downsampler is the scaled version

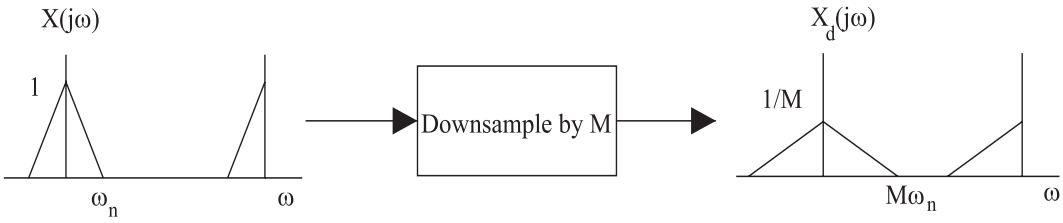


Figure 2-29: The discrete-time Fourier transforms of the input and output signals related by a downsampler.

(both in amplitude axis and frequency axis by a factor of M) of the DTFT of the input [25]. The inverse operation of downsampling is interpolation and it has the input/output DTFT relationship as in Figure 2-30.

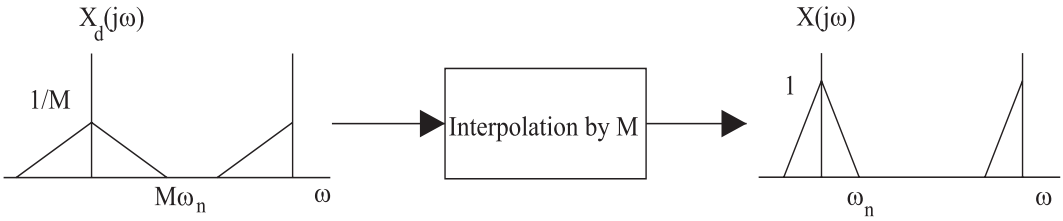


Figure 2-30: The discrete-time Fourier transforms of the input and output signals related by an interpolator.

Now the answer to the question of convergence is clear. If the downsampling factor is too high (e.g. $M\omega_n \geq \pi$), then the spectra of the downsampled signal will overlap and it will be impossible to recover the original signal, no matter what type of interpolation method is used. This effect is called aliasing, see Figure 2-31. In real world applications, the idealized bandwidth of a signal does not exist, thus the error due to aliasing is inevitable

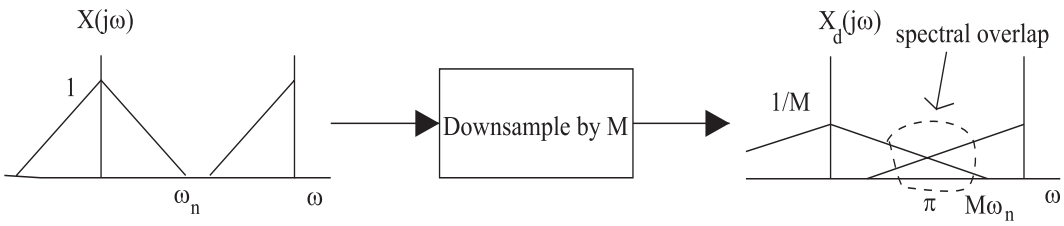


Figure 2-31: Aliasing effect and spectral overlap. The original signal is corrupted and cannot be recovered.

and the accuracy of the downsampling and interpolation process (Figure 2-28) depends on

the relative significance of the high frequency components of the original signal, as discussed in Subsection 2.4.3. Now suppose that the downsampling is acceptable in that the aliasing is within the tolerance. Let's study the interpolation process in the frequency domain, which will be proved rewarding. There is a uniform representation of the various kinds of interpolation in the frequency domain, provided that the interpolation step is fixed. The process is shown in Figure 2-32. The upsampling in Figure 2-32 is the following process in

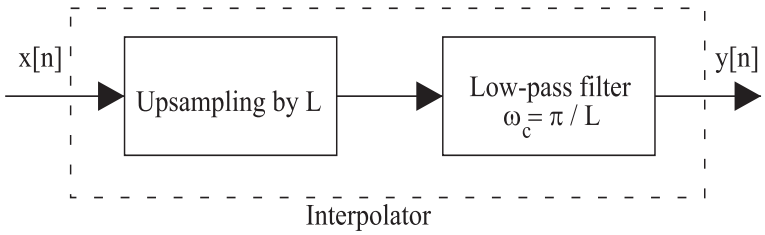


Figure 2-32: Block diagram of an interpolator. The upsampling is a process such that L - 1 zeros are inserted between every sample of the input signal x[n].

(2.36):

$$y[n] = \begin{cases} x[\frac{n}{L}] & n \in \{0, \pm L, \pm 2L, \ldots\} \\ 0 & \text{elsewhere.} \end{cases} \tag{2.36}$$

Evidently (2.36) is independent of the interpolation method. The quality of the interpolator then lies in the low-pass filter that follows. The example of the linear interpolator that has been given in Figure 2-12 in Section 2.3 clearly shows that linear interpolation is far from ideal and the quality of the interpolation is shown in Figure 2-27 (a). One can then imagine that the low-pass filter corresponding to the cubic spline should be much better. While the analysis from the treatment of a time sequence (e.g. cubic spline algorithm) to its frequency domain representation is revealing, the study in the opposite direction provides even more. That is, if the downsampled signal is processed by the procedure in Figure 2-32, then the associated effect in the time domain will be interpolation. Here the low-pass filter can be any one of the common approximations to the ideal low-pass filter, namely IIR or FIR filters. FIR filters will be better a choice in that the distortion induced by filtering will be relatively milder than that induced by IIR filtering [25]. In fact, there is a command in MATLAB (`interp.m`) that designs the filter and implements the filtering efficiently. After all the considerations above, it can be concluded that the accuracy of interpolation hinges on two factors: the aliasing due to downsampling and the proximity

of the low-pass filter to the ideal one. In other words, the worse of these two factors will determine the quality of the downsampling and interpolation (the recovery of the original response from its downsampled version) as in Figure 2-28.

Now that there are interpolation methods resulting from time domain and frequency domain perspectives, let's study the efficiency of these methods. It is important since if the additional cost imposed by interpolation is more expensive than what can be achieved by downsampling, then the downsampling and interpolation strategy will not be justified. There is no need to compare all methods, though, since the comparison between the fastest candidates should tell the winner. The candidates chosen are linear interpolation from time domain and low-pass filter interpolation from frequency domain (the filter length of this method is 4 by default. See [20]). The testing is to record the time to interpolate time sequences of different original lengths (10^5, 5×10^5, 10^6 and 5×10^6) and the downsampling (interpolation) factor is 4. The testing result is given in Figure 2-33. Frequency domain

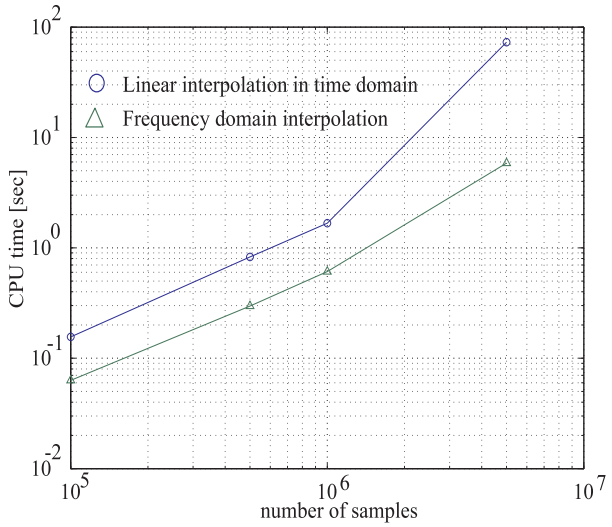


Figure 2-33: Computation time for linear interpolation and low-pass filter interpolation in logarithmic scale. Circle: CPU time by time domain method. Triangle: CPU time by frequency domain method.

method proves to be a more efficient method than any time domain method (linear, cubic spline, etc.), especially for long time sequences.

As a conclusion of the study on interpolation methods, let's try a numerical experiment and compare the accuracy of various kinds of interpolation methods. In this experiment, time sequences ($\cos(2\pi f \mathrm{t})$ where f is a variable) are downsampled (by factors of 2 and 4)

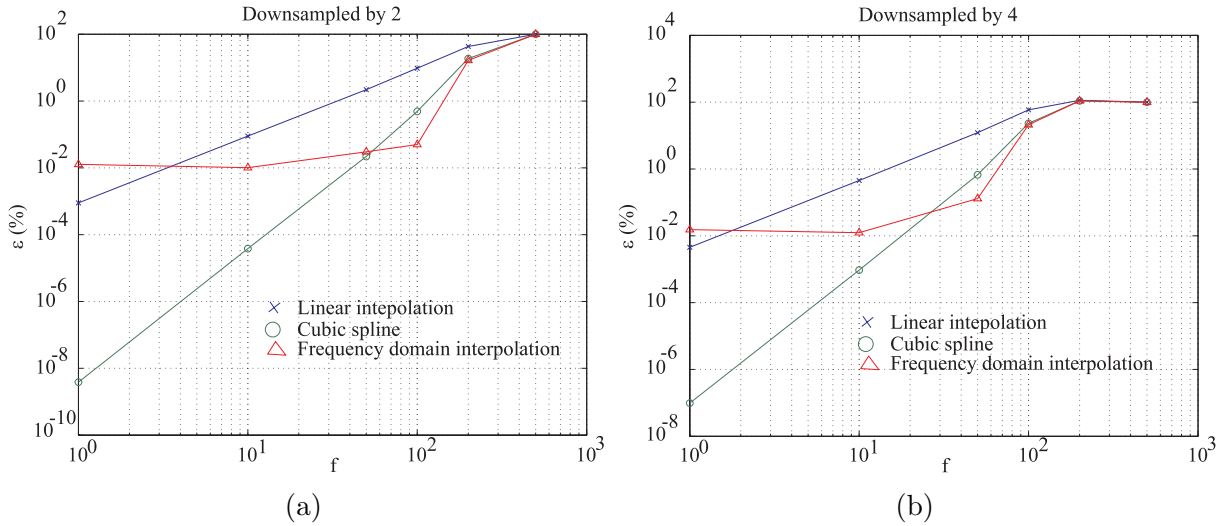


Figure 2-34: Percentage point-to-point error of different kinds of interpolation methods. The original signal is a sinusoidal signal sampled at 1000 Hz. (a) Effect in the case where downsampling factor is 2. (b) Effect in the case where downsampling factor is 4. Logarithmic scale.

and then interpolated by linear, cubic spline and low-pass filtering interpolation methods. The original downsampling rate of the time sequences is 1000 Hz. The error is defined as the percentage point-to-point difference between the original signal and the reconstructed signal as follows:

$$\varepsilon_{p2p} = \frac{\mathrm{E}[|\hat{y} - y|]}{\sqrt{\mathrm{E}[y^2]}},$$

where y is the original signal and $\hat{y}$ is the reconstructed signal. The results are given in Figure 2-34. Of the three methods considered, linear interpolation is the worst in that the error increases rapidly (e.g. $\epsilon_{p2p} > 1\%$ when $f > 10$ Hz) as the frequency of the original signal increases. There are some difficulties in choosing between cubic spline and low-pass filtering interpolation, though. The accuracy of cubic spline is obviously superior, especially in the case of low frequency signal interpolation. However, the computational cost for cubic spline is too high for it to be practical, especially for simulating MIMO systems with very long input sequence. In that case, the program will spend more time to interpolate than the time saved by downsampling. While the low-pass filtering interpolation method has worse quality compared with cubic spline, the actual value of error ($\frac{1}{100}\%$) is still acceptable[25]. After trading accuracy and efficiency, the choice is made to use low-pass filter interpolation.

[25]Further study of the filter should improve the accuracy.

Superposition

Having obtained the full length responses of the subsystems, it is finally time to superpose them to form the total response. This procedure is allowed by the linearity property of LTI systems of interest.

Some comments on feedthrough (i.e. $y = Du$) are appropriate at this time. Feedthrough causes a little trouble because the sum of the subsystem responses due to feedthrough is not the response of the original system due to feedthrough, but rather N times it, where N is the number of subsystems. A work around is to set the D-matrix of the subsystems to zero and finally add it back after all the subsystems have been simulated.

2.4.6 Section Summary

This section elaborated on the implementation issues considered in each of the functions in the flow chart, Figure 2-4 in Section 2.2. In the next section, examples will be given to show the results of `new_lsim.m` compared to those of `lsim.m`.

2.5 Simulation results and CPU time

2.5.1 Simulation results

In the section, examples will be given to show the application of `new_lsim.m`. The first example is the simulation of a SISO system. The specifications are as follows. The system simulated is made up of the flexible modes of SIM model v2.2 [29]. The input is one set of Magellan reaction wheel assembly[26] (RWA) disturbance data (4096 Hz $\times$ 210 seconds) [10]. Figure 2-35 is the result. The above simulation took about 73 seconds (compared with the result by (2.25), which is 75.14 seconds. Note also that downsampling effect is not important for SISO system simulation. See (2.25)). It should be noted that direct application of lsim.m fails because of the aforementioned memory problem. Note also that the response in Figure 2-35 (b) is not zero mean, while the input is zero mean, because the transient response of the system dominates during the 210 seconds considered.

Another experiment is done by simulating SISO systems of different sizes. This serves to show the CPU time comparison between `new_lsim.m` and `lsim.m` and the result can be seen in Figure 2-36. The systems have the size given in the x-axis and the input is

[26] Reaction wheel assembly is the primary source of disturbance on a space structure like a space-borne telescope.

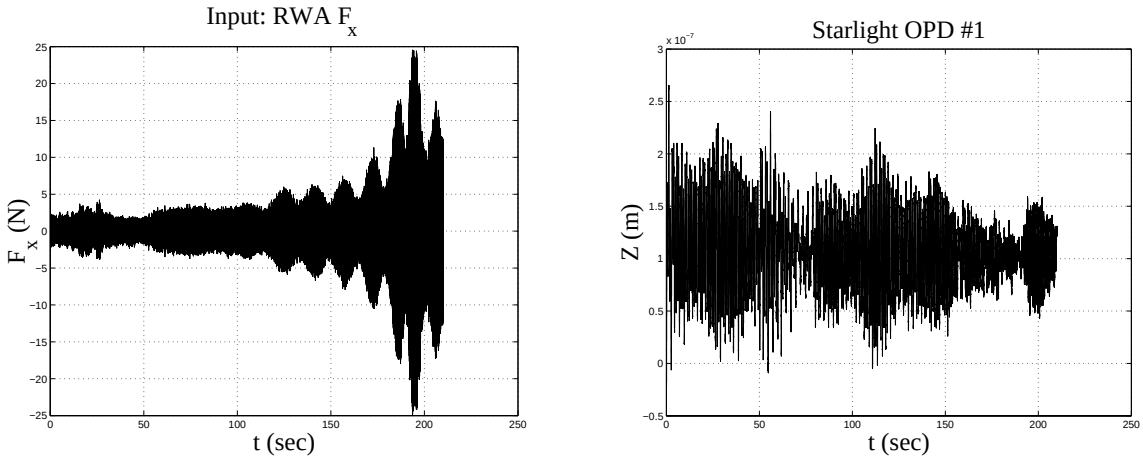


Figure 2-35: (a) Disturbance input (Magellan RWA F_X). (b) Performance output (Starlight OPD #1). Open loop simulation (no ACS and no optical control). Rigid body modes are removed.

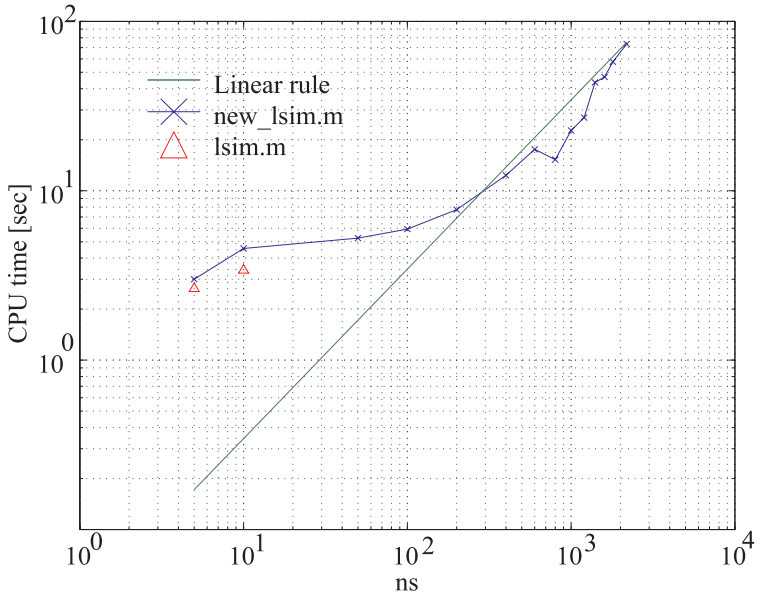


Figure 2-36: Simulation time for systems of various sizes. Line: linear time prediction. Line with star: recorded `new_lsim.m` time. Triangle: recorded `lsim.m` time.

$4096 \times 210 = 860160$ samples long. This input length is considered to be long. The simulation time does not include the time to diagonalize because the original system can be transformed to a diagonal one by indexing, see Appendix A. The result shows that `new_lsim.m` is slower than `lsim.m` when the problem is small (i.e. 5 state variables, 10 state variables. This is consistent with the inefficiency of small systems found in Subsection

2.4.2.). However, `new_lsim.m` can simulate systems with much higher dimensionality and thus extend the scope of the application of the simulation design method. This is consistent with the theme of this thesis. Also note that the computation time for `new_lsim.m` converges to the linear prediction law when system order is large. That is, the linear time property is preserved by `new_lsim.m`.

Another experiment was conducted in which randomly generated stable modal (block diagonal) systems (1000 state variables) with different numbers of input and output channels were simulated with randomly generated inputs (10^5 samples). The results are shown in Figure 2-37. Words of caution should be voiced, though, that the region of linear com-

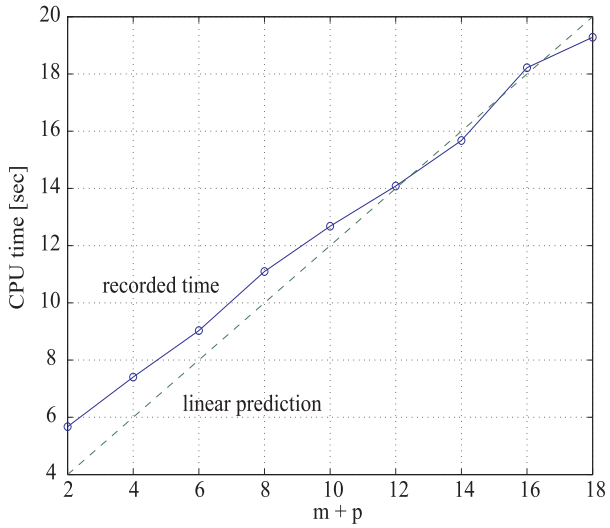


Figure 2-37: Simulation time for systems with various number of input and output channels. m and p are the numbers of input and output channels respectively. Linear prediction law is again given.

putation time is limited. For example, the time to simulate a small system (e.g. 5 state variables) or a system with many input/output channels (e.g. 20 inputs/20 outputs) will not be linear because of considerations that are neglected in the linear law.

In order to show that `new_lsim.m` is not just a desperate choice because `lsim.m` fails, yet another CPU time comparison is given in Figure 2-38. The systems simulated are randomly generated by MATLAB command `rss.m` and the input is 32768 sample length and randomly generated. The initial conditions are also randomly generated. It is clear from Figure 2-38 that `new_lsim.m` shows its value when the systems get large. Up to now, the computation time is given by specific examples. A simplified computational cost estimation

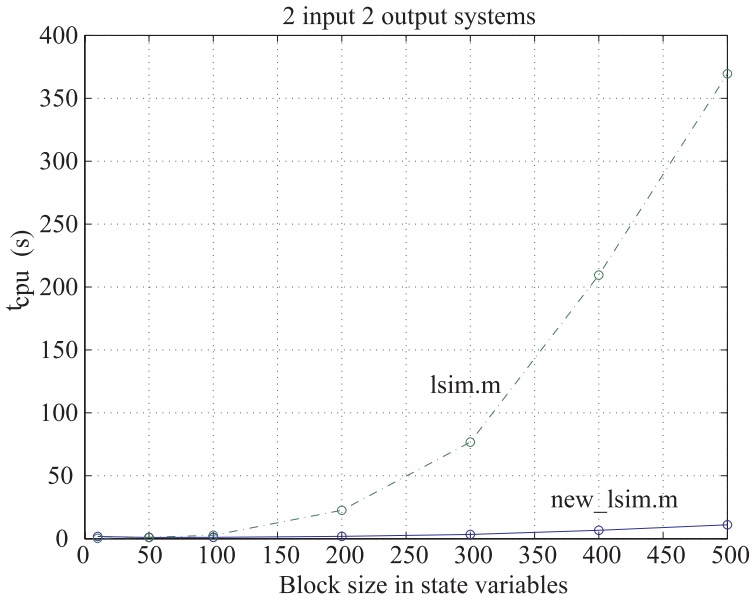


Figure 2-38: Simulation time of randomly generated systems. Solid line: `new--lsim.m` recorded time. Dash dot line: `lsim.m` recorded time.

is preferred so that the required time can be predicted before the simulation is carried out. Also, this estimation can serve as the quantitative criterion for the crossover at which the use of `new_lsim.m` is justified, as the issue is raised in the beginning of this chapter.

2.5.2 Computational time

Let's begin with the computation cost estimation for the standard state transition method (e.g. `lsim.m`). As discussed in Chapter 1, the cost for the state transition method can roughly be divided into two parts: discretization and state transition as the direct method shown in Figure 2-39. The main cost for the discretization is the matrix exponential (as discussed in Chapter 1), which essentially is the implementation of a Padé approximation. Due to the restriction of accuracy, the squaring of the intermediate matrix exponential, which accounts for most of the computational effort, is necessary. It can be verified that the FLOPS for computing a matrix-matrix product is $O(n^3)$, where n is the number of columns (or rows) of the matrix. However, the number of squarings required by the Padé approximation depends upon the specific case (i.e. the norm of the matrix). That is, although it is clear that the cost for discretization is $O(n^3)$, the exact coefficient in front of the cubic term is not fixed. An empirical rule has been found though, by taking the average

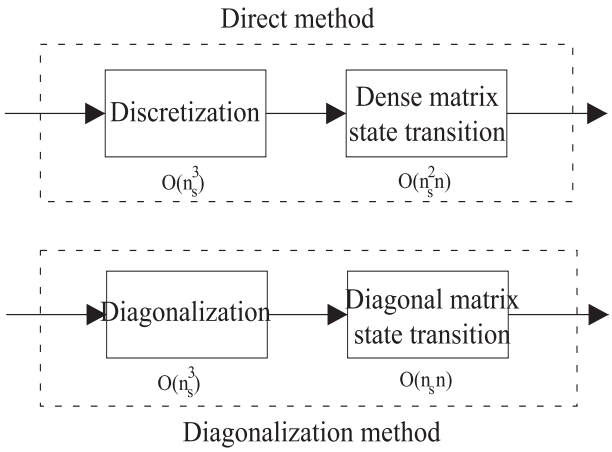


Figure 2-39: Schematics of direct state transition method and diagonalized state transition method. n_s is the number of state variables. n is the number of samples.

of the results from different runs and it is given as

$$\text{FLOPS} = 17.5742 \times n_s^3, \tag{2.37}$$

where n_s is the number of state variables, or in terms of CPU time, (2.37) becomes

$$T_{cpu} = 17.5742 \times n_s^3 \times \Delta t, \tag{2.38}$$

where Δt is the average time to compute one FLOP. The other part of the computational cost for the direct state transition method is that for the dense matrix state transition. Similar to the way the linear time law (2.25) is obtained, the CPU time prediction for dense matrix state transition is

$$T_{cpu} = 2 \times (n_s + m + p) \times n_s \times n \times \Delta t, \tag{2.39}$$

where n_s is the number of state variables, m is the number of input channels, p is the number of output channels, n is the number of samples, Δt is the average time for computing one FLOP and the factor of 2 is due to the fact that matrix multiplication includes scalar multiplication and addition. Combining (2.38) and (2.39) the time estimate for direct state transition is

$$T_{direct} = 17.5742 \times n_s^3 \times \Delta t + 2 \times (n_s + m + p) \times n_s \times n \times \Delta t. \tag{2.40}$$

Now let's turn to the computational cost for the diagonalized state transition method (e.g. `new_lsim.m`), as shown in Figure 2-39. The cost for computing the similarity transform

in `ss2mod71.m`, in which most of the computational effort is dedicated to computing the eigenvalues and eigenvectors, is given by (2.23) in Subsection 2.4.1 and its CPU time version is

$$T_{cpu} = 27.8276 \times n_s^3 \times \Delta t. \tag{2.41}$$

Apart from the cost for diagonalization, there are other costs such as discretization and state transition. However, due to the block diagonal structure of the A-matrix, the computational cost for discretization is very cheap and can be ignored. The cost for block diagonal state transition has been given in (2.25) and thus the time estimate for diagonalized state transition is

$$T_{diag} = 27.8276 \times n_s^3 \times \Delta t + 2 \times (2 + m + p) \times n_s \times n \times \Delta t. \tag{2.42}$$

Analyzing (2.40) and (2.42) the following points can be made:

- The costs for the two state transition methods are both $\mathrm{O}(n_s^3)$. However, the components that contribute to the cost are different. In the direct method, it is discretization and in the diagonalized method, it is diagonalization. Therefore, if the original system is discrete-time, the direct method might be better. On the other hand, if the original system is in modal form (e.g. structural dynamics problems), then the diagonalized method is better since there is no need to compute the diagonalization.

- The state transition cost for the direct method is $\mathrm{O}(n_s^2)$, while that of the diagonalized method is just $\mathrm{O}(n_s)$. Consequently, the advantage of `new_lsim.m` over `lsim.m` will be more obvious when the number of samples required to be processed is large.

- In the above discussion, downsampling was not included. If downsampling is employed, then the computational cost is reduced accordingly.

- Care should be taken in interpreting the estimate due to (2.40) and (2.42) because the average computational time per FLOP can be different due to factors like machine type, current workload, computer adaptability[27] and so on. For example, the computation time per FLOP for state transition was found to be 5×10^{-9} seconds, but a more accurate time per FLOP for the eigenvalue problem and matrix exponential is 2×10^{-9} seconds on the current test machine (see Chapter 1 for specifications). In

[27] Different computers perform different tasks with different efficiency. For example, computer A can solve systems of linear equations faster than B does, but B can compute eigenvalues faster than A.

addition, the relationship between FLOPS and CPU time might not be as simple as one is related to the other by a factor of Δt. For example, in MATLAB, while the CPU time/FLOPS relationship for matrix exponential, diagonalization and diagonalized state transition (DTSS) is predicted quite accurately by (2.40) and (2.42), the CPU time relationship for the case of dense matrix state transition (`ltitr`) is not as simple. (e.g. For a SISO system with 400 state variables and the sample length is 32768, `ltitr` took 213 seconds (on the test machine) to solve the problem, while (2.39) says the estimate is 52.69 seconds if Δt is taken to be 5×10^{-9} seconds.) Nevertheless, the application of the time (FLOPS) estimators (2.40) and (2.42) is not restricted to `ltitr` only and have value in general.

2.6 Accuracy Issues and Error Analysis

Now that we have demonstrated significant improvements in computation time, an important question remains unanswered. Are the simulation results produced by `new_lsim.m` accurate? The question is to be answered in this section. First let's make some definitions. The reference, or the true value of any simulation result is defined to be the response by standard MATLAB routine `lsim.m`. Denote the response by `lsim.m` y, and that by `new_lsim.m` $\hat{y}$. In addition, let's define error metrics

Point-to-point error $\varepsilon_{p2p} = \frac{\mathrm{E}[|\hat{y}-y|]}{\sqrt{\mathrm{E}[y^2]}}$, where E is the average operator. This metric provides information on the proximity of the two waveforms.

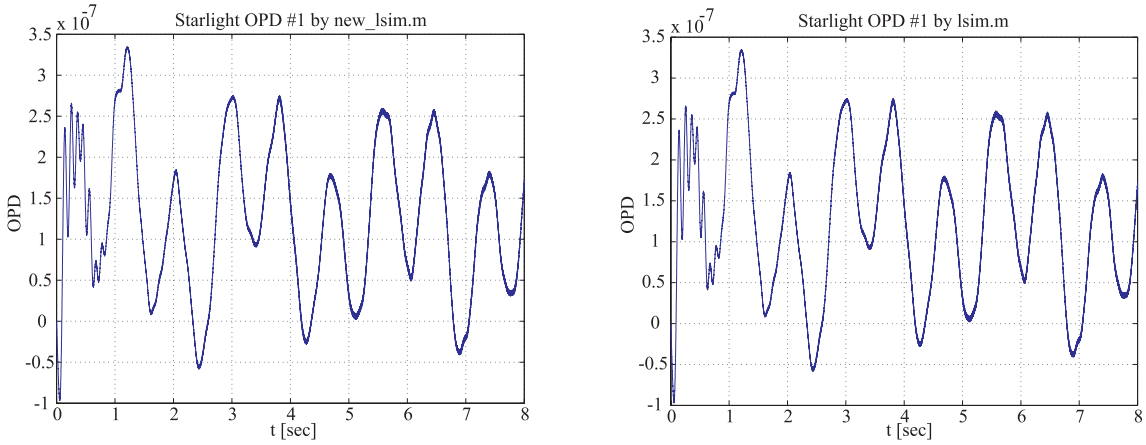
Relative mean error $\varepsilon_m = \frac{|\mathrm{E}[\hat{y}]-\mathrm{E}[y]|}{|\mathrm{E}[y]|}$, where E is the average operator. This metric gives information on accuracy regarding mean statistics.

Relative RMS error $\varepsilon_\sigma = \frac{|\sqrt{\mathrm{E}[\hat{y}^2]}-\sqrt{\mathrm{E}[y^2]}|}{\sqrt{\mathrm{E}[y^2]}}$, where E is again the average operator and this metric gives statistical accuracy regarding root-mean-squared (RMS) values.

2.6.1 Simulations and Accuracy

Again a simulation experiment is carried out to see how well $\hat{y}$ approximates y. The settings are, a 100 state variable SIM model (open loop and rigid body modes removed) and 32768-samples Magellan RWA input $\mathrm{F_X}$ data with a sampling rate of 4096 Hz. The output is starlight OPD # 1. The reason for these specific settings is to keep the simulation time by `lsim.m` reasonable. Figure 2-40 shows that the responses by `new_lsim.m` and `lsim.m` look

alike and Figure 2-41 is the plot of $\epsilon \triangleq \hat{y} - y$. Similar experiments were conducted with



(a) (b)

Figure 2-40: (a) Response by `new_lsim`. (b) Response by `lsim.m`.

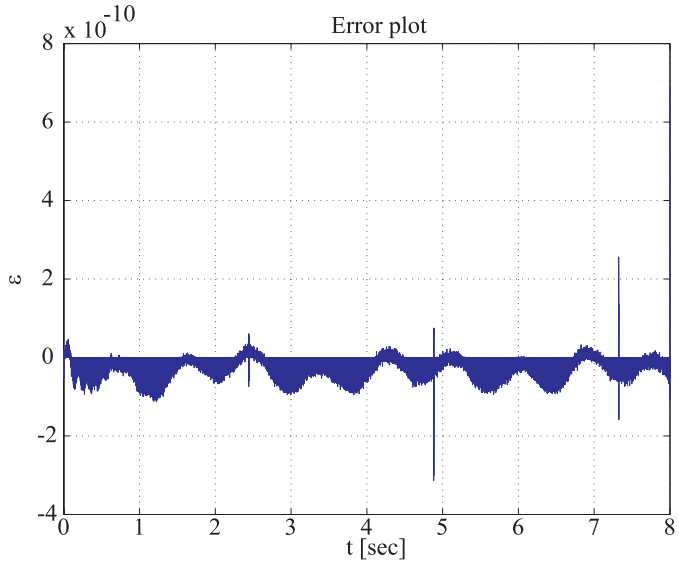


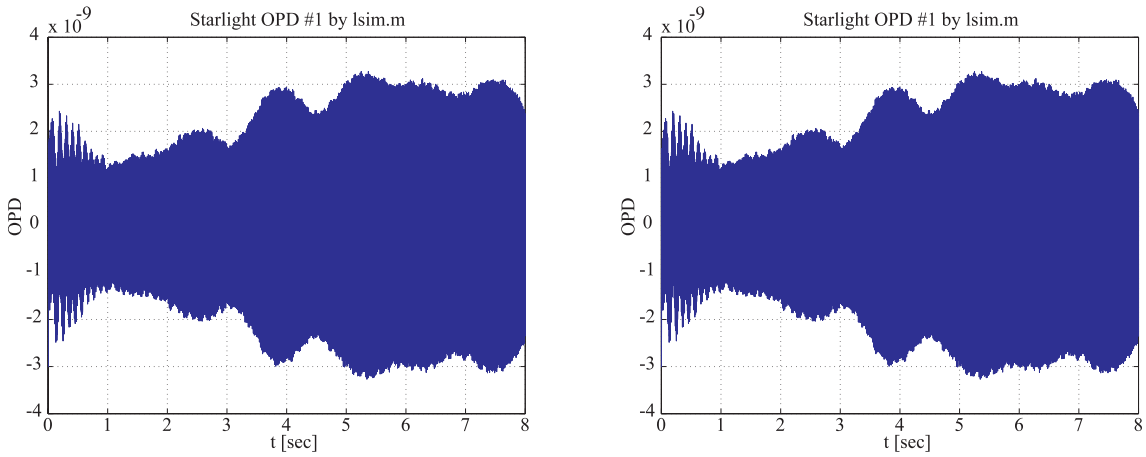
Figure 2-41: Error plot. $\hat{y} - y$

different sets of RWA disturbance data and the results are summarized in Table 2.9. For all the experiments above, the downsampling factor was set to 4, even though downsampling might not be beneficial for the SISO case, because the goal of these experiments is to see how the error comes about when downsampling is employed. It should be pointed out, though, that downsampling shows its value for MIMO cases, otherwise it will be redundant. Finally, another experiment was conducted in which the open loop SIM model was cascaded

Table 2.9: Point to point error, relative mean and relative RMS error

RPM	0–100	100–400	400–700	700–1000	1000–1500	1500–2000
$\epsilon_{p2p}(\%)$	0.0150	0.0147	0.0152	0.0146	0.0153	0.0158
$\epsilon_m(\%)$	0.0172	0.0172	0.0172	0.0172	0.0172	0.0172
$\epsilon_\sigma(\%)$	0.0172	0.0171	0.0172	0.0171	0.0172	0.0172
RPM	2000–2500	2500–3000	3000–3500	3500–4000	4000–4500	
$\epsilon_{p2p}(\%)$	0.0152	0.0154	0.0147	0.0156	0.0155	
$\epsilon_m(\%)$	0.0172	0.0172	0.0172	0.0172	0.0172	
$\epsilon_\sigma(\%)$	0.0171	0.0171	0.0171	0.0172	0.0172	

with an optical control system (high-pass filter). This experiment is interesting in that the accuracy requirement is more stringent due to the more significant proportion of the high frequency components. Here the downsampling factor is set to 4 manually[28]. The settings in this experiment are the same as those in the previous one, except that a high-pass filter is cascaded with the original plant. The responses by `lsim.m` and `new_lsim.m` are shown

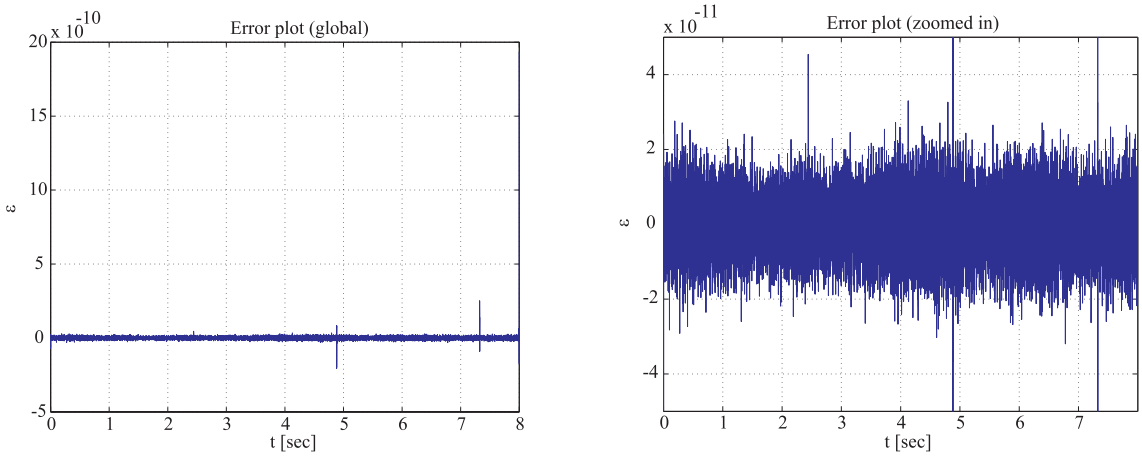


(a) (b)

Figure 2-42: (a) Response by `new_lsim`. (b) Response by `lsim.m`. Open loop plant with optical control.

in Figure 2-42. Figure 2-43 shows the error. The relative point-to-point error in this case is 0.2936 % because the strong low frequency signal was suppressed by the high-pass filter (compare with Figure 2-40). Note also that the error in the beginning and at the end is much greater because of the transient response of the low-pass filter implemented by the interpolation (It is a drawback of the low-pass filter interpolation method.)

[28] `new_lsim.m` tries to estimate the appropriate downsampling factor before a simulation begins and the estimate for this case is 1.



(a) (b)

Figure 2-43: Difference between responses by `new_lsim.m` and `lsim.m`. (a) Global plot. (b) Zoomed in y plot.

2.6.2 Error Analysis

Let's begin the error analysis with the schematic of the sources of error. Recall in Subsection 2.4.3 that there are two ways to implement downsampling and they lead to Figures 2-44 and 2-45. It should be clarified that Figure 2-44 and 2-45 are not the usual signals and

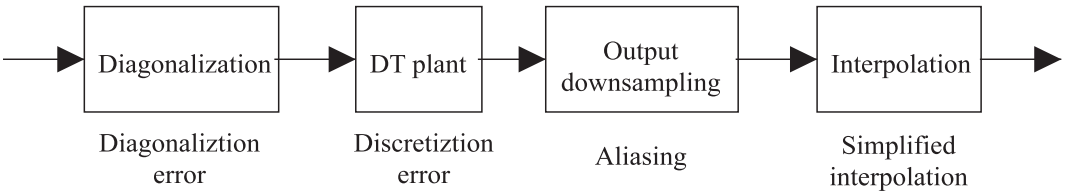


Figure 2-44: Equivalent sequence of processes and the corresponding sources of error. Output downsampling scheme.

systems block diagrams, but rather schematics of the ordered processes that can introduce error (compare with Figure 2-4[29]). In the development that follows, only Figure 2-44 will be focused on since it is the method that is adopted by the current version of `new_lsim.m`. Corresponding to Figure 2-44 there are several points to elaborate:

- **Finite machine precision**. Truncation (or rounding depending upon the kind of scheme adopted by the machine) error increases with the number of operations. That

[29] Also note that Figure 2-44 is not the true implementation either. It is equivalent to the actual output downsampling scheme in the sense that the errors are introduced in the same way. Figure 2-44 is adopted because it illustrates the concept better.

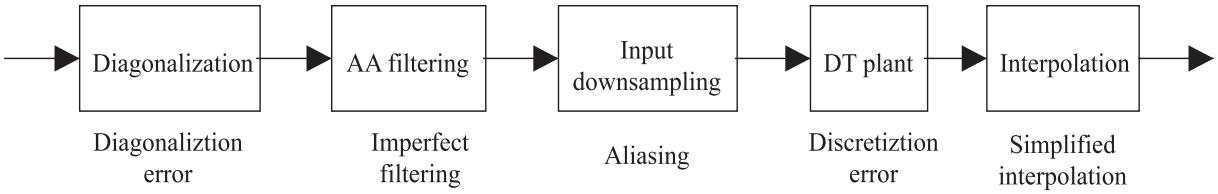


Figure 2-45: Sequence of processes and the corresponding sources of error. Input downsampling scheme.

is, the more computation the algorithm has, the more likely that the final error is bigger.

- **Diagonalization error**. It is known that numerical computation of eigenvalues and eigenvectors can sometimes be unstable in that the results can be highly ill-conditioned. Nevertheless, there are many cases where the problem is well-posed. As a fair treatment, an error estimate of 10^{-13} is added to account for the general degradation due to diagonalization. Fortunately for the current case, the magnitudes of the signals range from 10^{-7} to 10^{-9} and the diagonalization error should not cause too much trouble.

- **Discretization error**. Recall in Section 2.3 that the construction of the discretized system (the derivation of the formulae for A_d, B_d, C_d and D_d) is based on certain assumptions. Specifically, if the zero order hold discretization scheme is used, then the assumption is inaccurate unless the input signal is piecewise constant (e.g. step function or zero input). If first order hold discretization scheme is employed, then the signal must be piecewise linear (e.g. ramp function) in order to fit the formula exactly. See [3] for some discussion on this issue. However, it should be pointed out that the reference here is the state transition simulation method (i.e. `lsim.m`), which also makes the assumption such as ZOH or FOH. Thus if the same discretization scheme and discretization time step are used by `new_lsim.m`, then there should be NO difference (or error) between y and $\hat{y}$. Another subtlety arises in the case of the input downsampling scheme (Figure 2-45) though, in that the discretization error does exist. The reason is as follows: Even if it is true that the input signal is constant (for ZOH) over one time step, there is no guarantee that it is constant for more than one step, which is the time step for the downsampled simulation.

- **Aliasing due to downsampling**. This is the first serious source of error in that it can actually cause unsatisfactory results. To understand how the error comes about and the way to estimate it, let's study the processes of downsampling and interpolation in Figure 2-46 again. It is clear from Figure 2-46 that even if the ideal interpolator

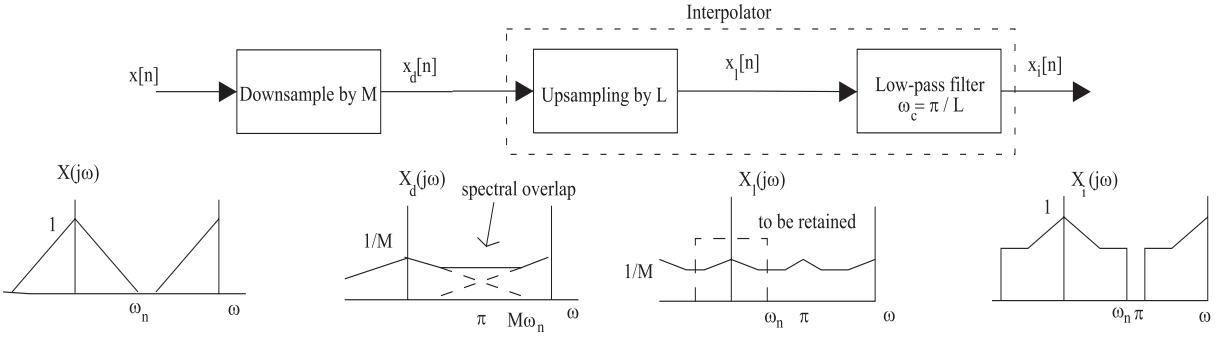


Figure 2-46: The processes of downsampling, interpolation and the intermediate spectra.

is used, the reconstructed signal, $x_i[n]$, is not the same as the original signal $x[n]$ because signals in real world applications are not ideally bandlimited. The relative significance of the overlapping part of the spectrum determines the quality of the $x_i[n]$. Therefore, a reasonable error estimate should take into account the information of the signal part and the overlapped part of the spectrum. An example of this error estimate can be the ratio between a bound on the high frequency part of the signal (to be overlapped by downsampling) and the RMS value of the overall signal[30]. If this ratio is close to 1, then the degradation due to downsampling is expected to be high. On the other hand, if the ratio is small, then aliasing will not be the main problem of downsampling. Then the question is: How do we find this ratio before the output is actually computed? The answer is as follows. For the computation of the bound on the high frequency signal, the relationship between input and output spectra, (2.26) can be used.

$$|Y(j\omega)| = |H(j\omega)U(j\omega)| \leq |H(j\omega)||U(j\omega)|.$$

Here the bound is approximated by $|Y(j\omega)|$, $|U(j\omega)|$ can be approximated by the maximum magnitude value of the input spectrum, which can be efficiently computed by FFT. $|H(j\omega)|$ can be obtained by one of the standard frequency response routines.

[30]The estimate obtained from the ratio between maximum value and RMS value is more conservative than that obtained from the ratio between RMS value and RMS value.

As for the RMS value of the signal, Parseval's theorem [27] can be applied.

$$\begin{aligned}\int_{-\infty}^{\infty} |y(t)|^2\, dt &= \frac{1}{2\pi}\int_{-\infty}^{\infty} |Y(j\Omega)|^2\, d\Omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} |H(j\Omega)U(j\Omega)|^2\, d\Omega,\end{aligned}$$

where $H(j\Omega)$ and $U(j\Omega)$ have already been obtained. Ω is angular frequency in rad/sec. The reader might ask: Why bother computing all these frequency domain quantities? The answer is that the spectra and frequency response can be computed efficiently, especially for large-order systems with special structure like block diagonal A-matrix. Now let's try to apply these tools to estimate the error in the first experiment (Figures 2-40 and 2-41). Figure 2-47 shows the spectrum of the input signal. It

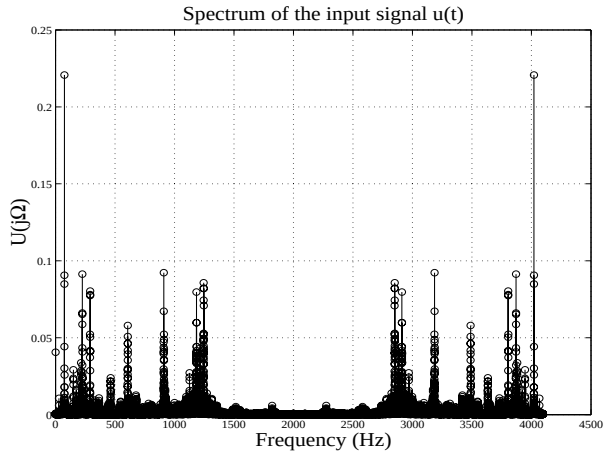


Figure 2-47: Spectrum of the input signal $u(t)$ (Magellan RWA disturbance input).

should be pointed out that the input signal is treated as deterministic, even though it is stochastic in nature (i.e. a realization of a stochastic process is deterministic). From Figure 2-47, it is clear that the approximate $|U(j\Omega)|$ can be chosen as 0.1. For the purpose of illustration, the Bode diagram of the plant is computed and is given in Figure 2-48. The frequency point highlighted in Figure 2-48 deserves some elaboration. Since the original sampling rate is 4096 Hz and the downsampling factor here is 4, the frequency components of the output signal beyond $\frac{1}{4} \times \frac{4096}{2} = 512$ Hz will be aliased. The most conservative choice for the frequency response $|H(j\Omega)|$ should then be at 512 Hz, which has the value of -210 dB. Therefore, the bound for the high frequency signal is $0.1 \times (-210 \text{ dB}) \approx 3.162 \times 10^{-12}$. The RMS value of the signal can

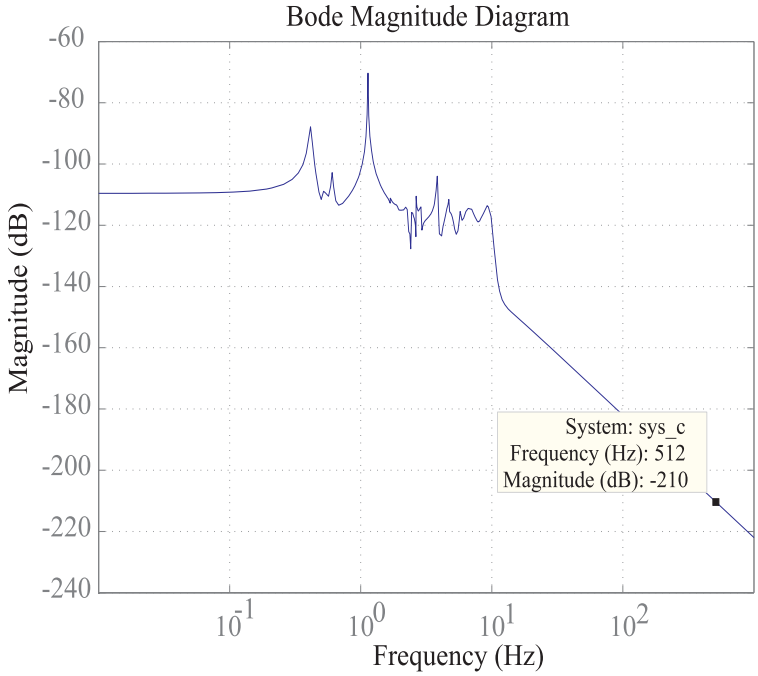


Figure 2-48: Bode diagram of the system. The highlighted point (512 Hz) is used to estimate the error due to aliasing. The downsampling factor is 4 and the original sampling rate is 4096 Hz.

be obtained by applying Parseval's Theorem and the value is 1.6244×10^{-7} (compare with 1.4681×10^{-7}, the value obtained by actually taking the RMS of the output $y(t)$). With all the ingredients for the error estimate, it's finally time to see if aliasing causes the error. The estimate is calculated to be 0.0019%, which is lower than the actual error level ($\approx 0.01\%$). Now it is clear that aliasing is not the major source of error. Therefore, interpolation is expected to be responsible for the error, as will be shown in the next point.

- **Interpolation error**. The accuracy issues regarding interpolation have been studied in Subsection 2.4.5 and shall not be repeated here. Nevertheless, as the continuation of the problem (experiment 1 error analysis) in the previous point, a simple experiment is to be conducted to show the effect of interpolation error. Recall in Subsection 2.4.5 that cubic spline is very accurate for cases where aliasing is not a problem. Therefore, the error due to cubic spline interpolation should be small, as can be verified by Figure 2-49 and the relative point-to-point error is 0.0037%, which is much closer to the 0.0019% estimate of aliasing error found previously. This simple calculation

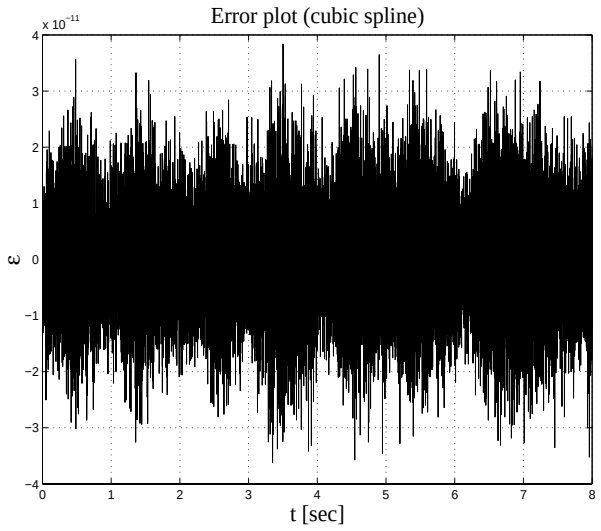


Figure 2-49: Error plot between the original signal and the cubic spline reconstructed signal. Note that the y-axis is scaled by 10^{-11}.

shows that interpolation by low pass filtering is the main cause of error in this problem. Finally, it should be pointed out that the accuracy issue does not warrant the use of cubic spline since it is too expensive.

After considering the most obvious sources of error (due to output downsampling scheme), it can be concluded that the error is mainly due to aliasing and imperfect interpolation. This fact shows the direction to error prediction and possible improvement to better accuracy in a future upgrade.

2.7 Chapter Summary

In this chapter, issues concerning the simulator `new_lsim.m` were discussed in detail. In Section 2.1, the characteristics of the targeted problems were discussed briefly. The problems that warrant the use of `new_lsim.m` should be large-order in terms of the number of state variables, input/output channels and samples. Also, it was pointed out that the simulator was particularly suited for structural dynamics problems in that the problems oftentimes are in the required modal forms. Then, in Section 2.2 the flow chart of the simulator was given, followed by a brief discussion for each of the processes. After that, in Section 2.3 the basics for the manipulations of `new_lsim.m` were studied. The formula (2.8) for state transition (or discretization) with zero order hold was derived. Then the benefits

for diagonalization, namely fast discretization, decoupling and linear time state transition, were discussed. At the end of Section 2.3, the concept of frequency domain representation of discrete-time signals (sequences) was introduced and the properties (aliasing and bandwidth) that are important to the following implementations were discussed. In Section 2.4, detailed discussion of each of the implementations shown in Figure 2-4 was given. Modal decomposition, or diagonalization (implemented by `ss2mod71.m`) was first discussed and the related topic of the computation cost for the eigenvalue problem was briefly touched. Then the issues regarding subsystem planning were elaborated. This subsection can be divided into three main parts: sampling rate considerations, block size considerations and the discussion of the subroutine `seg_plan_x.m`. After the study on subsystem planning, the issues with downsampling were discussed. There two downsampling schemes were suggested (output downsampling and input downsampling). Output downsampling was the focus since it was adopted by the current version of `new_lsim.m` and the corresponding efficiency and accuracy issues were studied. The discussion of downsampling was then followed by a brief talk on the discretization scheme. In the next subsection (Subsection 2.4.5), the details on simulation solvers (`ltitr` and DTSS) were given and then the accuracy and efficiency issues with interpolation were addressed. After all the details on the implementations, simulation examples were given in Section 2.5 and the computation cost (FLOPS and CPU time) was discussed briefly. In the last section (Section 2.6) of this chapter the accuracy issues of `new_lsim.m` were addressed and the error estimate was proposed.

In Chapter 3, the application of `new_lsim.m` will be extended to the problems with feedback control systems.

Chapter 3

Closed Loop System Simulation

In this chapter, the issue of closed loop system simulation will be discussed. The block diagrams of the general problem settings are in Figure 3-1. In the diagrams, w is the external

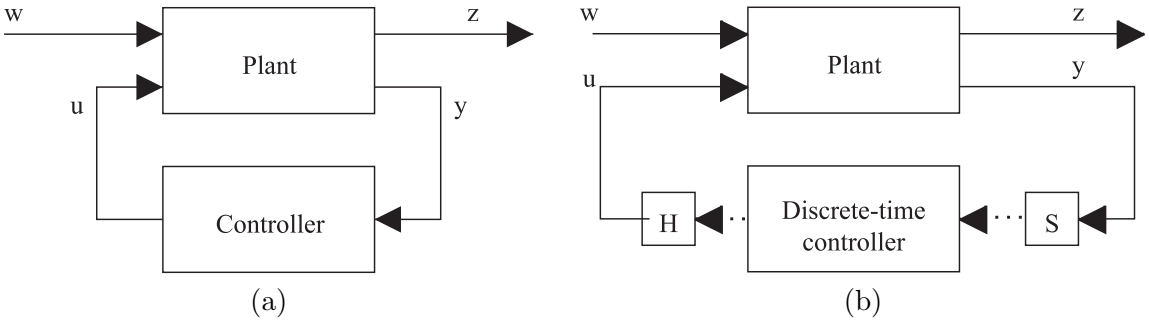


Figure 3-1: Block diagrams of a feedback control system. (a) Continuous-time (analog) controller. (b) Discrete-time (digital) controller. S denotes a sampler and H denotes a holder. Solid lines are continuous-time signals. Dotted lines are discrete-time signals. Note also that these quantities can be scalars or vectors.

disturbance input, z is the performance output, y is the measurement and u is the control input. S is the mathematical symbol of a sampler and its real world counterpart is the series of a zero order hold (ZOH), A/D converter and a (binary) coder. H denotes the holder and it is usually a D/A converter followed by a ZOH. The structure in Figure 3-1 (b) will be studied primarily since digital control is more common and powerful than analog control today. The combination of a continuous-time plant and discrete-time (digital[1]) control system is generally called a sampled-data control system (or hybrid system). Feedback control is introduced for a variety of reasons: Stabilization, performance, and resistance to uncertainties are the most obvious of them. For an excellent reference on feedback control

[1]Digital and discrete-time signals are the exactly the same in that digital signals are discrete in both time and amplitude, while discrete-time signals have continuous values at discrete time instants.

systems, see [19]. The introduction of one or several feedback controllers does increase the challenge to the analysis of the model as will be discussed immediately. The problem can be further complicated by the dimensionality of the plant, which always seems to grow faster then available computation ability. Consequently, a method of simulating the large-order hybrid system is desired.

3.1 New Problems Induced

In order to apply standard LTI system simulators, it is necessary to reshape the systems in Figure 3-1 into the standard form as in Figure 2-1 in Chapter 2. This processing is straightforward for ordinary size systems but it might constitute some problems for large-order system simulations. For example, if the plant in Figure 3-1 has its A-matrix block diagonal, then the A-matrix of the autonomous system will not generally have this nice feature as explained below. If the plant is represented in state space form[2]

$$\left[\begin{array}{c|cc} A & B_w & B_u \\ \hline C_z & D_{zw} & D_{zu} \\ C_y & D_{yw} & 0 \end{array}\right] \tag{3.1}$$

and the controller has the following state space form

$$\left[\begin{array}{cc} A_k & B_k \\ C_k & D_k \end{array}\right], \tag{3.2}$$

where subscripts w and u denotes quantities related to disturbance input and control input respectively, subscripts z and y are related to performance output and measurement output and subscript k denotes quantities related to the controller, then the autonomous system has the state space form

$$\left[\begin{array}{cc|c} A + B_u D_k C_y & B_u C_k & B_u + B_u D_k D_{yw} \\ B_k C_y & A_k & B_k D_{yw} \\ \hline C_z + D_{zu} D_k C_y & D_{zu} C_k & D_{zw} + D_{zu} D_k D_{yw} \end{array}\right]. \tag{3.3}$$

The off-diagonal terms, $B_u C_k$ and $B_k C_y$ in the block A-matrix of (3.3) are not expected to be zero, otherwise there will be no control effect at all. Figure 3-2 shows the structures of the A-matrices with and without control. Since these cross terms essentially couple

[2]The D_{yu} is missing here to avoid an algebraic loop, i.e. the coexistence of feedthroughs in the plant and the controller.

the dynamics of the autonomous system[3], the open loop simulator, `new_lsim.m` cannot be applied directly unless a diagonalization is again conducted. This will be discussed later in this chapter.

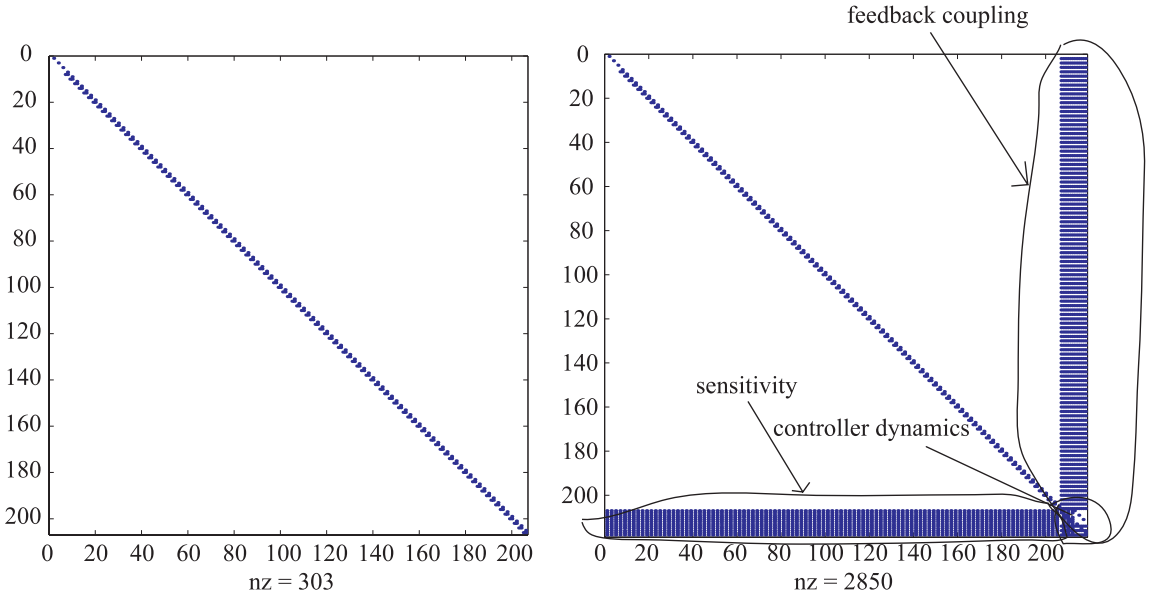


Figure 3-2: (a) Structure of the A-matrix of the open loop system. (b) Structure of the A-matrix of the closed loop system. Typically the number of controller state variables is significantly less than the number of plant state variables.

The other problem is that the autonomous system is hybrid (containing continuous-time and discrete-time state variables[4]) and in the context of computer simulation, the system can have multiple sampling rates involved and thus results in a time-variant system. In order to see why, it should be noted that what a computer simulation actually implements is a scheme like Figure 3-3 because a digital computer cannot handle real continuous-time sequences. If the sampling rates of the samplers in the outer loop and inner loop are the same, let the plant and controller absorb the samplers and holders and then the autonomous system is still time-invariant as in Figure 3-4. However, if the sampling rate of the controller (inner loop) is very low[5] relative to the plant dynamics, for example, 1Hz, then with the

[3]This coupling has detrimental effects on control as well. In structural control problems, a controller designed to stabilize some modes can destabilize other modes not considered. This effect is called spillover and is one of the motivations for modal sensor technology.

[4]Hybrid systems, in general, mean something even more complicated in that discrete states (e.g. logic, event), as well as continuous states (e.g. current, voltage) coexist in the systems considered. Automata, as well as conventional tools such as differential equations, are required to analyze them. In the contexts of this thesis, however, hybrid merely means the combination of continuous-time and discrete-time state variables.

[5]The sampling rate of a digital controller is usually determined when the controller is designed. A rule

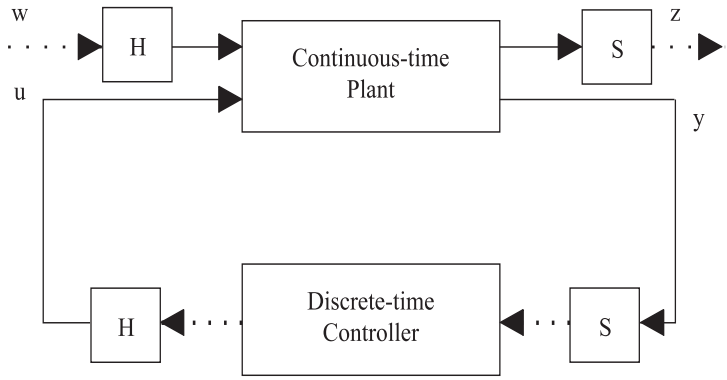


Figure 3-3: Computer simulation of a sampled-data control system. Solid lines are continuous-time signals. Dotted lines are discrete-time signals.

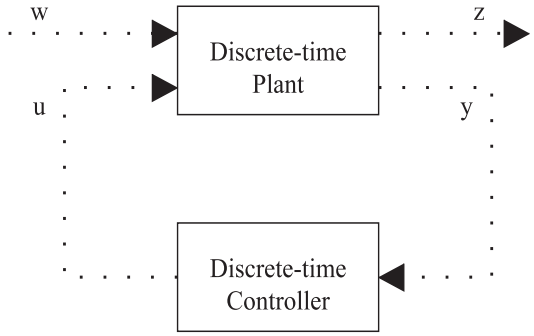


Figure 3-4: Single sampling rate autonomous system. Sampling rate is that of the controller.

1Hz sampling rate the output sequence z in Figure 3-3 cannot in general reflect the true behavior of the original continuous-time system. For example, if one tries to design a $\mathcal{H}_2$ controller [38] and it has good disturbance rejection for the case in Figure 3-4, when it is applied to the original system in Figure 3-1 (a), the performance is not necessarily good because of the large inter-sample intervals. Figure 3-5 illustrates the problem. Therefore, a better scheme is to discretize the plant at a higher sampling rate as in Figure 3-6. If the samplers and holders in Figure 3-6 are absorbed then the time-variant autonomous system in Figure 3-7 results. The system in Figure 3-7 is time-variant in that the controller acts like a holder and isolator at each fast sampling instant except slow sampling instants, but at every slow sampling instant it is an ordinary discrete-time dynamical system. Therefore,

of thumb is that it should be 10 times the plant bandwidth. It is bad for the sampling rate to be too low, but it is also harmful to have it too high since the feedback will introduce too much delay.

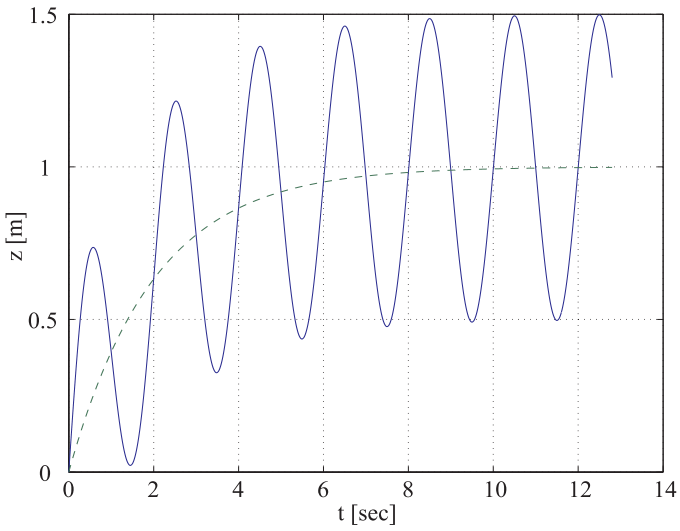


Figure 3-5: Discrepancy between actual response and the desired response. Solid: actual response. Dashed: desired response. The sampling rate is 1 Hz.

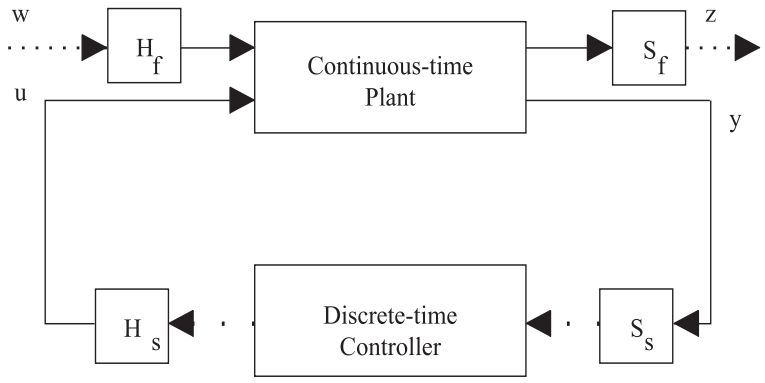


Figure 3-6: Fast discretization scheme for simulation. Subscript f and s denote fast and slow sampling rates respectively. Dotted lines with different densities denote discrete-time signals sampled at different rates.

the whole system is time-variant.

To sum up, the introduction of feedback digital controllers induces couplings and multiple sampling rates. This poses a new challenge for large-order systems simulations since the methods from open loop simulation (Chapter 2) cannot be applied without additional work.

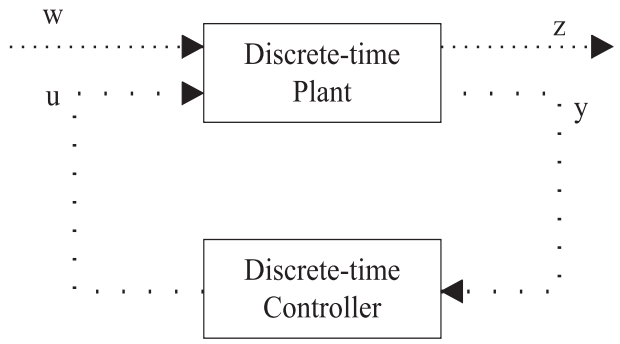


Figure 3-7: Autonomous system with two sampling rates. The dotted line with more dots is the discrete-time signal with higher sampling rate.

3.2 Proposed Solutions to the New Problems

In this section the solutions to the two simulation problems will be discussed. First, two methods for decoupling are suggested. Then, two methods for dealing with multiple sampling rates are suggested. The details and analysis of these approaches will be deferred to Section 3.3.

3.2.1 Solutions to Dynamics Coupling

In discussing dynamics coupling, let's for the moment assume that the controller is continuous-time[6]. If the autonomous system is diagonalizable, then the procedure for open loop system simulation can be applied with a rediagonalization. It should be pointed out that a sufficient for diagonalizability of a matrix is that all its eigenvalues are distinct. This is generally true for a closed loop system since its eigenvalues are moved to the desired locations, which are usually distinct, by feedback control. For example, the purpose of the attitude control system on a spaceborne structure is to stabilize the rigid body modes (i.e. poles clustered at the origin). The result of the stabilization is to move the associated eigenvalues farther to left half s-plane in case of a continuous-time system or inside the unit circle in the case of a discrete-time system. This effect essentially relieves the problem of defectiveness of the A-matrix.

The second method is heuristic in that some of the couplings between the plant and controller dynamics are ignored if they are deemed weak. Whether the coupling is weak or

[6]One of the most common digital controller design (synthesis) techniques is to first design the desired analog controller and then find its digital approximation by methods like bilinear (Tustin) transformation.

not is decided on an ad-hoc basis. There is some discussion for testing and accuracy of this approximation in Section 3.3. An example application is concerned with the attitude control system of a space telescope. Here the rotational attitude angle measured by a gyroscope[7] is the linear combination of both the rigid body modes and the flexible modes dynamics. However, these measurements (attitude angles) are dominated by the rigid body modes (e.g. the measurement error due to ignoring the flexible modes in SIM v2.2 is less than 0.01%). Therefore, it is acceptable to make the approximation that the measurements are obtained by sensing only the rigid body modes and this means that the columns in C-matrix due to flexible modes can be set to zero as shown in Figure 3-8. This assumption is convenient

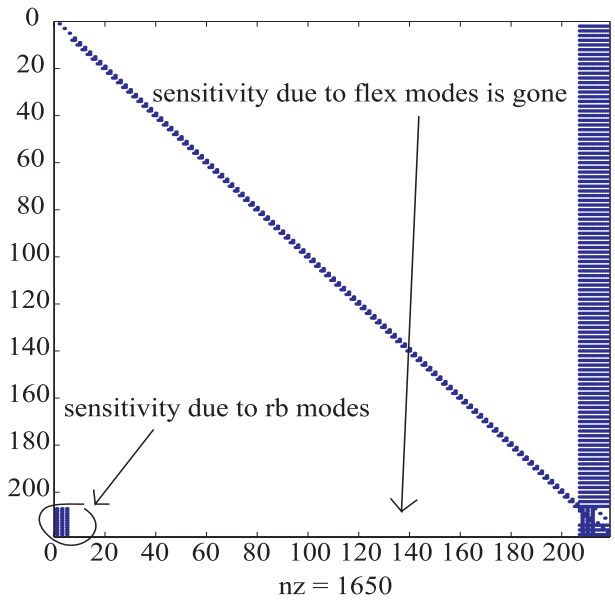


Figure 3-8: The effect of flexible modes on controller dynamics is ignored.

in that it allows the problem of a closed-loop system simulation to be treated sequentially, namely a small closed-loop system with rigid body modes is first simulated exactly and then the large-order flexible mode system can be simulated with the calculated control input in an open-loop fashion. Figure 3-9 shows the idea of this decoupling graphically and the simulation scheme is shown in Figure 3-10. This scheme has the advantage that the large-order subsystem is not changed due to the change of the controller due to control tuning. In the context of `new_lsim.m`, the rediagonalization of the large-order subsystem is

[7]Gyroscope measures angular rate only, the attitude angles are actually obtained by integrating the angular rate.

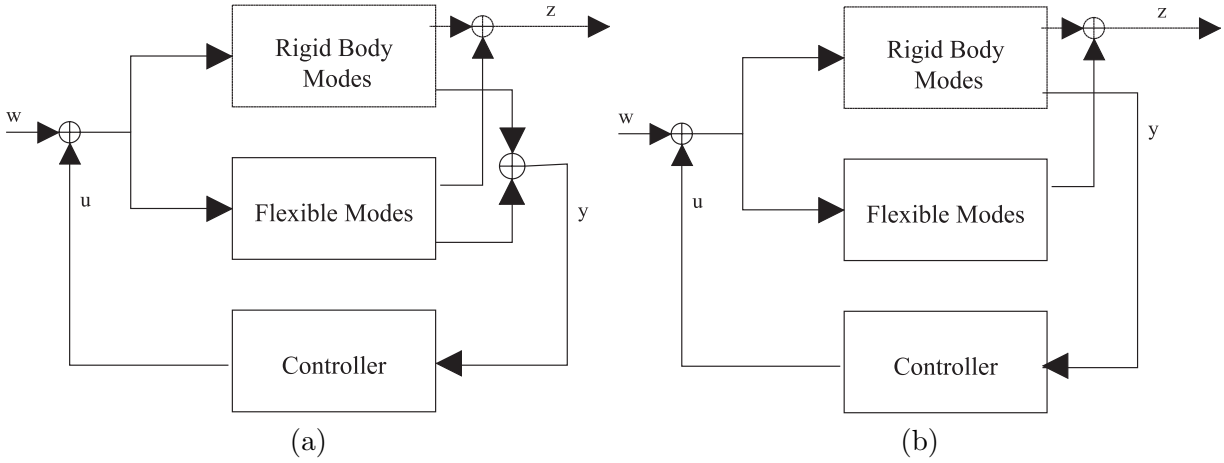


Figure 3-9: (a) A system with rigid body and flexible dynamics coupled. (b) The coupling is ignored with acceptable approximation.

avoided. The accuracy of this approach is determined by the weight of the selected modes on the measurement. For example, in the 200-mode SIM v2.2 problem, the results by rediagonalization scheme and forced-decoupling scheme (only rigid body modes are assumed to be sensed) have a point-to-point difference of no greater than 0.002% when the external input is the Magellan RWA disturbance.

3.2.2 Solutions to Multiple Sampling Rates

Two methods are proposed here. The philosophy of these treatments is to transform the multirate system into a single rate system so that it can be handled with tools that already exist. Essentially, either a slow sampling rate is converted to a fast rate or vice versa. Method 1 transforms the discrete-time controller into a continuous-time one and then discretizes it with the fast sampling rate. Specifically, the procedure is to let the discrete-time controller in Figure 3-1 (b) absorb the sampler and holder so it becomes a continuous-time system. The resultant system configuration is in Figure 3-11. Now the continuous-time system can be discretized again at a single sampling rate. This method, in essence, is the resampling of the discrete-time controller. Approximations must be made while using this method. Fortunately, when the sampling rate of the controller is not too low, the error due to resampling is acceptable. This issue will be discussed in Section 3.3.

Method 2 works in the opposite direction of Method 1 in that the resultant system

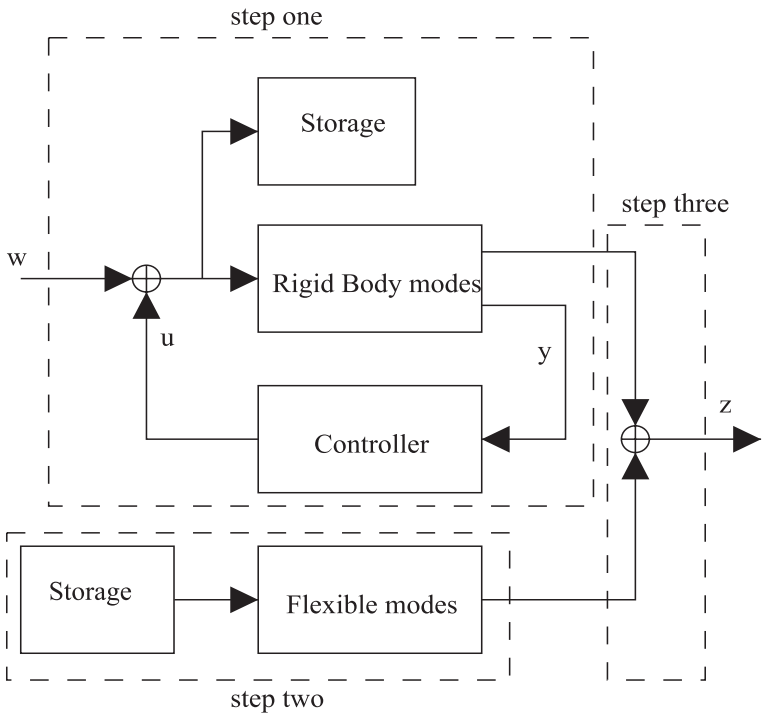


Figure 3-10: Three step simulation scheme of a forced decoupled system.

is sampled at a slow sampling rate[8]. The procedure that allows this treatment is lifting [33], which basically trades the temporal dimension (length of samples) with geometrical dimension (number of inputs and outputs) via the application the of state transition formula. While Method 2 is exact in that no approximation has been made, it suffers from the problem that discretization for a large A-matrix can take a lot of time. Only in the case of very low controller sampling rate (e.g. less than 1 Hz) should Method 2 be the choice. However, in some special cases where the controller is sensitive to only a few modes (i.e. the measurement is the linear combination of a few modes), the sequential simulation scheme discussed in Subsection 3.2.1 can be applied and only the small closed-loop system is simulated with Method 2. The situation is shown in Figure 3-12 and it can be exact or obtained from the procedure described in Subsection 3.2.1. However, the exactness of Method 2 is obviously lost.

[8]Method 2 requires that the fast sampling rate is an integer multiple of the slow sampling rate.

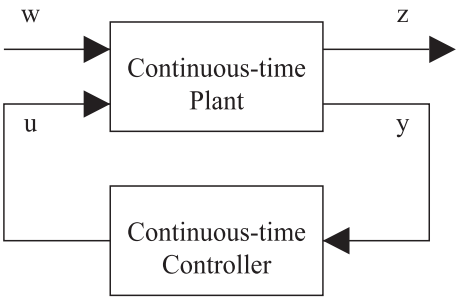


Figure 3-11: Method 1. Converting the discrete-time controller into a continuous-time controller.

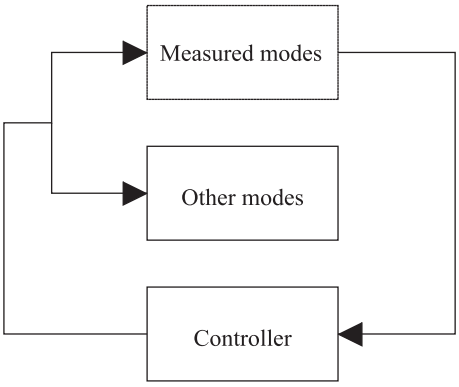


Figure 3-12: Situation in which a sequential procedure can be applied.

3.3 Basis and Details for the Manipulations

The notion of diagonalization has been discussed throughout in Chapter 2 and will not be repeated here. Therefore, the first topic in this section is the issue with the heuristic decoupling method in Subsection 3.2.1. The second topic is the implementation and accuracy of the continuous-time approximation of a discrete-time controller and the notion of lifting follows.

3.3.1 Forced Decoupling Approximation and Error Bound

This method is good since the computation due to controller redesign is avoided. However, it suffers from the risk that the simulation will be unrealistic when the contribution to the measurement of the ignored modes is significant. Although it is the performance output that is being concerned, it is justified to check only the measurement since the closed loop system is assumed to be stable and thus the error shall not grow. A few ways are proposed here as

tests to see if the decoupling makes sense and the computation effort of these tests increases in order. The problem block diagram is in Figure 3-9 (a), where the partial system with flexible modes is assumed to be stable and block diagonal. The principle of these methods is to check the ratio between the magnitude related quantities[9] of the measurement signals due to ignored and kept modes. In other words, the measurements of the two systems in step one and step two in Figure 3-10 (the measurement of the flexible mode subsystem is not shown in the figure) are processed to make the decision.

Method 1 is based on the following fact,

$$\|H(s)\|_2^2 = \sum_{i=1}^{m} \|G\delta_i\|_2^2, \tag{3.4}$$

where $\|H(s)\|_2^2$ is the square of the 2-norm[10] of the transfer function matrix of the system, $\|G\delta_i\|_2^2$ denotes the square of 2-norm of the output (can be MIMO) of the system due to impulse at input channel i and m is the number of input channels. The method first computes the 2-norms of the transfer function matrices of the rigid body mode and flexible mode subsystems and then weighs them with the RMS values of the external input signal to estimate the output signal RMS value by the idea of (3.4). After that, these quantities are compared to see the contribution of the flexible modes to the measurement. Perhaps the following formula should make the statement above more precise.

$$E_i = \frac{\sum_{j=1}^{m} H_{ij}^f W_j}{\sum_{j=1}^{m} H_{ij}^r W_j} \qquad \forall i \in \{1, 2, \ldots, p\}. \tag{3.5}$$

Here E_i is the weighting, if it is small, then the forced decoupling is justified. H_{ij}^f denotes the 2-norm of the transfer function of the flexible mode subsystem, from input channel j to output channel i. H_{ij}^r denotes something similar, except that it's for the rigid body mode subsystem. W_j is the RMS value of the j^{th} input. m is the number of output channels and p is the number of input channels. It is not a problem to obtain H_{ij}^r since the dimension of the closed loop rigid body mode subsystem is small. Routines like `norm.m` should be sufficient for this purpose. The computation of H_{ij}^f is more involved if `norm.m` is directly applied. Fortunately, the following procedure makes the computation efficient. If the system has

[9] For energy signals, the test quantity is energy. For power signals, the test quantity is power.

[10] There is another popular choice of norm, namely ∞-norm, but the computation of it for large-order system is not easy.

state space form

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right],$$

and the transfer function matrix is $H(s)$, then

$$\|H(s)\|_2^2 = \text{trace}\{CLC^{\mathrm{T}}\},$$

where

$$L = \int_0^\infty e^{A\tau} BB^T e^{A^T\tau} d\tau$$

is the controllability grammian[11] [32] and it can be obtained by solving the following Lyapunov Equation

$$AL + LA^T + BB^T = 0.$$

The bottleneck of computing the 2-norm of the transfer function matrices is transferred to the computation of the solution to the Lyapunov Equation. However, for block diagonal systems there exists a fast Lyapunov Equation solver `newlyap.m` [6]. With all these formulae and tools it is possible to roughly estimate the significance of the ignored modes (flexible modes) to the measurement. The reason why this is just a rough estimation is that the estimated output RMS thus obtained is often overestimated. However, it gives the user confidence of forced decoupling if the conclusion from this test is promising. Therefore, method 1 provides a quick check of the level of approximation.

Method 2 is a semi-brute force approach. It estimates the measurement magnitudes due to rigid body mode and flexible mode subsystems by the definition of the input-output frequency response

$$|\,Y(j\Omega)| = |\,H(j\Omega)W(j\Omega)|\,, \tag{3.6}$$

where $|\,Y(j\Omega)|$ denotes the magnitude of the Fourier transform of the measurement, $H(j\Omega)$ denotes the frequency response (or transfer function) of the system and $W(j\Omega)$ is the Fourier transform of the external input, which is usually obtained by Fast Fourier transform (FFT) numerically. The error index can then be

$$E = \frac{|\,Y_{flex}|}{|\,Y_{rigid}|},$$

where $|\,Y_{flex}|$ and $|\,Y_{rigid}|$ are the bounds for flexible modes response and rigid body modes response respectively. The frequency response of the rigid body mode subsystem can be

[11] This form is for continuous-time systems.

obtained by any standard routine without any problem since it is small. The frequency response of the large-order flexible, however, requires some care in computation. However, if the A-matrix is block diagonal, then the computation burden can be relieved significantly by exploiting the block diagonal structure as follows

$$
\begin{aligned}
H(s) &= C(sI-A)^{-1}B + D && (3.7)\\
&= \begin{bmatrix} C_1 & C_2 & \cdots & C_N \end{bmatrix} \begin{bmatrix} X_1 & 0 & \cdots & 0 \\ 0 & X_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & X_N \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix} + D && (3.8)\\
&= C_1X_1B_1 + C_2X_2B_2 + \cdots + C_NX_NB_N + D, && (3.9)
\end{aligned}
$$

where $X_i = (sI - A_i)^{-1} \quad \forall\, i \in \{1, 2, \ldots, N\}$ and the sizes of the block entries can be formed with liberty. However, by trial and error an advantageous block size has been discovered to be about 40 state variables and the realization of this computation is written in the code `fast_fr_x.m`. The magnitude of the measurement can then be obtained by evaluating the integral (IFT)

$$
\begin{aligned}
|\, y(t)| &= \left| \frac{1}{2\pi} \int_{-\infty}^{\infty} Y(j\Omega) e^{j\Omega t} d\Omega \right| && (3.10)\\
&\leq \frac{1}{2\pi} \int_{-\infty}^{\infty} \left|\, Y(j\Omega) e^{j\Omega t} \right| d\Omega && (3.11)\\
&\leq \frac{1}{2\pi} \int_{-\infty}^{\infty} |\, Y(j\Omega)| \left|\, e^{j\Omega t} \right| d\Omega && (3.12)\\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} |\, Y(j\Omega)|\, d\Omega && (3.13)\\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} |\, H(j\Omega) W(j\Omega)|\, d\Omega. && (3.14)
\end{aligned}
$$

The last equality is due to (3.6). It should be noted that from (3.12) on the right-hand-side is a bound on the signal $y(t)$. There are other ways to make use of the information of the FFT of the input and system frequency response, though. One of the appropriate candidates will be Parseval's Theorem [27]. That is, for a signal $x(t)$

$$
\int_{-\infty}^{\infty} |\, x(t)|^2\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\, X(j\Omega)|^2\, d\Omega. \qquad (3.15)
$$

In either case, the evaluation of the integral can only be approximated for signals more complicated than those encountered in textbooks. There are numerous approaches to do

the integration: Riemann[12] sum (rectangular rule), trapezoidal rule, Simpson's rule, etc. In the spirit of fast estimation, the simplest form (i.e. Riemann sum) can be used. With these bounds obtained, the contribution of the ignored modes to the measurement can be estimated[13]. In conclusion, this method provides a more realistic picture of the level of approximation and the precision of the prediction can be adjusted by the integration scheme, which again is a tradeoff between accuracy and efficiency.

The third method is a complete brute force approach. In this method, the actual rigid body mode subsystem and the open loop flexible mode subsystem are simulated and the measurements are obtained. Although it is brute force, it is not useless since conclusions can be drawn from some small-scale sample problems. For example, in case of MIMO systems, only the measurement output, which usually has a much smaller dimension than the performance output, is needed to make the decision. Also, the simulation run can be stopped once steady state is reached. This method serves as a final judgement for the use of the forced decoupling method suggested in Subsection 3.2.1.

As an example, a system with 3 rigid body modes (i.e. 6 associated state variables) and 200 flexible modes (6 inputs, 4 performance output and 3 measurement output) is made up and a continuous-time LQG controller is designed and this system is going to be simulated using the forced decoupling method discussed above. Before that, let's try the tests to predict the accuracy. The 2-norms of the rigid body mode and flexible mode subsystems are obtain and listed in Tables 3.1 and 3.2. The RMS values of the input to the six channels

Table 3.1: 2-norms of the rigid body mode subsystem.

Input channel	1	2	3	4	5	6
Output channel						
1	0.0000	0.0012	0.0041	0.0028	0.0003	0.0001
2	0.0026	0.0000	0.0033	0.0003	0.0064	0.0002
3	0.0036	0.0014	0.0000	0.0001	0.0002	0.0024

are 3.5878, 1.1564, 3.1779, 0.1668, 0.4644 and 0.0559. The estimated magnitude bounds and the corresponding error bounds are summarized in Table 3.3. Although the bounds are quite big, the actual level of error is expected to be lower since the 2-norm is usually more conservative.

[12]Georg Friedrich Bernhard Riemann, 1826–1866, was one of the most important mathematicians of all times. He made important contributions to geometry and complex analysis and his work laid the foundations

Table 3.2: 2-norms of the flexible mode subsystem. The actual values should be scaled by 10^{-3}.

Input channel	1	2	3	4	5	6
Output channel						
1	0.0005	0.0040	0.0055	0.0080	0.0306	0.0119
2	0.0114	0.0039	0.0317	0.0004	0.1711	0.0249
3	0.0066	0.0103	0.0035	0.0002	0.0229	0.0672

Table 3.3: Estimated output RMS values and error bounds given by computation of 2-norm and input RMS values.

Output channel	1	2	3
Rigid body modes	0.0151	0.0229	0.0149
Flexible modes ($\times 10^{-3}$)	0.0403	0.2270	0.0612
Error bound (%)	0.2660	0.9911	0.4103

Secondly, let's try using the frequency domain method, the integrals (3.14) are in Tables 3.4 and 3.5 and a more realistic error bound is obtained in Table 3.6. The results in row 1 and 2 in Table 3.6 are obtained from summing the rows in Table 3.4 and 3.5.

Table 3.4: Estimated magnitude bounds on the responses at each output channel due to each input of the rigid body mode subsystem.

Input channel	1	2	3	4	5	6
Output channel						
1	0.0000	0.0003	0.0035	0.0002	0.0000	0.0000
2	0.0014	0.0000	0.0026	0.0000	0.0000	0.0000
3	0.0023	0.0004	0.0000	0.0000	0.0000	0.0002

Finally, actual simulation is carried out to verify the bounds. The results summarized in Table 3.7. It is relieving that Tables 3.6 and 3.7 provide results very close to each other. As another verification, the four performance outputs are compared between the case of forced decoupling and rediagonalization and the point-to-point error is no greater than 0.01%.

of the theory of general relativity.

[13] In fact, `new_lsim.m` uses this method to estimate the error due to downsampling.

Table 3.5: Estimated magnitude bounds on the responses at each output channel due to each input of the flexible mode subsystem. Actual values should be scaled by 10^{-6}.

Input channel	1	2	3	4	5	6
Output channel						
1	0.0059	0.0167	0.0149	0.0137	0.0036	0.0012
2	0.0535	0.0078	0.2337	0.0004	0.0790	0.0032
3	0.0183	0.0165	0.0176	0.0004	0.0089	0.0119

Table 3.6: Estimated Magnitude bounds and error bounds given by frequency domain method.

Output channel	1	2	3
Rigid body modes	0.0041	0.0040	0.0028
Flexible modes ($\times 10^{-6}$)	0.0560	0.3777	0.0736
Error bound (%)	0.0014	0.0093	0.0026

Table 3.7: Output RMS values and error bounds given by simulation.

Output channel	1	2	3
Rigid body modes	0.0017	0.0019	0.0009
Flexible modes ($\times 10^{-6}$)	0.0096	0.1982	0.0210
Error (%)	0.0006	0.0103	0.0023

3.3.2 Continuous-time Approximation of a Discrete-time System

The original setting is in Figure 3-1 (b) and the continuous-time controller is obtained by combining the sampler, discrete-time controller and the holder. For the sake of simplicity, only the SISO case is discussed here. However, the principle naturally extends to the MIMO case. Let's denote the discrete-time controller K(z) and the converted continuous-time controller $K_{d2c}(s)$ in Figure 3-13. The accuracy of the method depends upon how close $K_{d2c}(s)$ approximates K(z). Now suppose that the input y(t) in Figure 3-13 is bandlimited (e.g. the sampling rate of K(z) is higher than the Nyquist rate.). This assumption will be relaxed later and the consequence will be discussed. Secondly it is noted that the processing of $K_{d2c}(s)$ is LTI. Finally let's assume (at least for the time being) that the holder is an ideal D/C converter that reconstructs a continuous-time signal from its samples exactly provided that the sampling rate is higher than the Nyquist rate. Under these assumptions, the

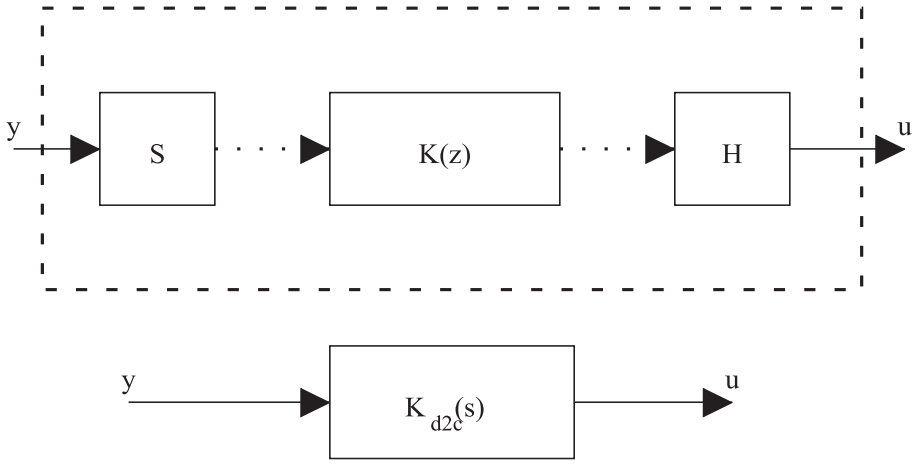


Figure 3-13: Schematic of the equivalent continuous-time controller.

processing HK(z)S can be regarded as the discrete-time processing of the continuous-time system $\mathrm{K}_{d2c}(\mathrm{s})$ [25] and has the relation

$$K_{d2c}(j\Omega) = \begin{cases} K(e^{j\omega})|_{\omega=\Omega T} & |\,\Omega| < \frac{\pi}{T} \\ 0 & |\,\Omega| > \frac{\pi}{T}, \end{cases} \tag{3.16}$$

where T is the sampling period, $K(e^{j\omega})$ is the frequency response of the discrete-time system and $K_{d2c}(j\Omega)$ is the frequency response of the equivalent continuous-time system. Now it is clear where the approximation occurs since it is not possible for the holder to be ideal. Often times the holder is a zero order hold and has the frequency response in Figure 3-14.

The problem is not finished yet because a way is needed to obtain $\mathrm{K}_{d2c}(\mathrm{s})$ in Figure 3-13. It is a common practice to discretize a continuous-time system into a discrete-time one as in Figure 3-15. In fact, the computer simulation process itself fits into the framework of Figure 3-15. Routines are available for implementing this discretization (e.g. `c2d.m` in MATLAB). On the other hand, the inverse of the process, which is defined as obtaining a K(s) such that its discretization is $\mathrm{K}_{c2d}(\mathrm{z})$, is also possible and routines are readily available. However, the continuous-time system obtained in this way is not the desired $\mathrm{K}_{d2c}(\mathrm{s})$ in Figure 3-13 since HS is not in general an identity. Figure 3-16 is an example in which the holder is ZOH. Again, the problem lies in the fact that the holder is not ideal. In fact, if the holder is ideal, then the processing in Figure 3-15 is the continuous-time processing of a discrete-time system and the two systems in it have the relation

$$K_{c2d}(e^{j\omega}) = K(j\Omega)|_{\,\Omega=\omega/T} \qquad |\,\omega| < \pi, \tag{3.17}$$

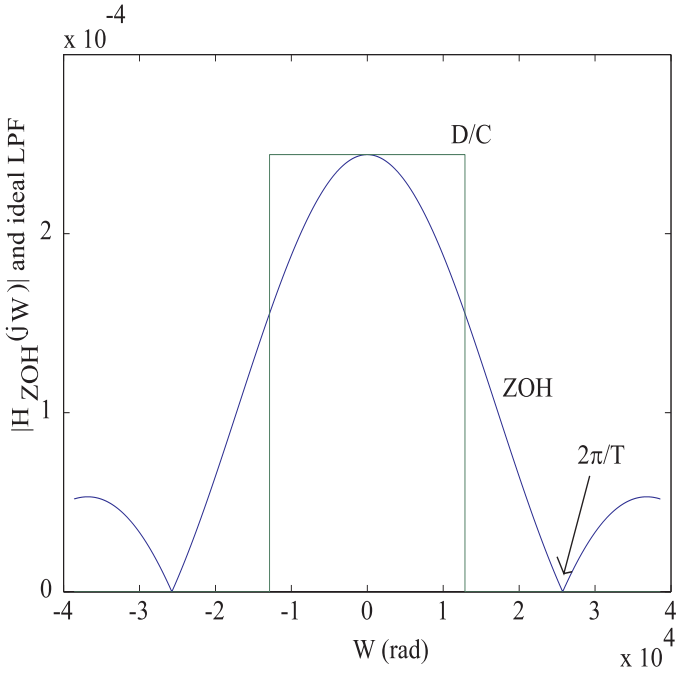


Figure 3-14: Frequency response of a zero order holder and the ideal reconstruction low-pass filter of an ideal D/C converter. T is the sampling period.

where T is the sampling period, $K(j\Omega)$ is the frequency response of the continuous-time system and $K_{c2d}(e^{j\omega})$ is the frequency response of the equivalent discrete-time system. If the nice assumptions are valid and (3.17) and (3.16) are combined, the desired discrete-time controller can be obtained. But the reality is that it is not possible. Then a practical question will be: Is it possible to improve the accuracy under the condition that the holder is not ideal? Recall from Section 2.4 in Chapter 2 that if the sampling rate of the controller is higher, the spectrum of the discrete-time signal is more concentrated at $0, \pm\pi, \pm 2\pi, \cdots$ and from Figure 3-14, one should expect the approximation converges to the ideal case as the sampling rate goes to infinity.

A simple example is given here to show the trend. A sampled-data control system (200-mode SIM v2.2) is stabilized by discrete-time controllers with different sampling rates and the MATLAB SIMULINK block diagram is shown in Figure 3-17. Here the controller (with analog anti-aliasing filter) is designed by standard LQR approach (e.g. MATLAB's `lqr.m`) and the external input is the Magellan RWA disturbance. The responses are starlight OPD #1,2,3,4 and they are compared with the response obtained by simulating an all continuous-time system (discrete-time controller is converted to continuous-time, see Figure 3-18) with

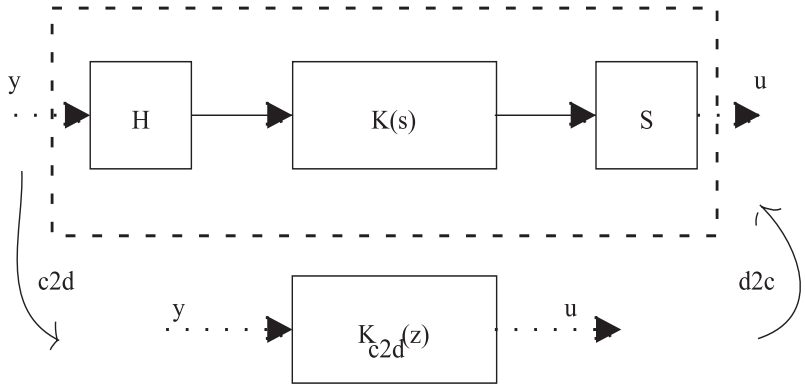


Figure 3-15: Schematic of the equivalent discrete-time controller.

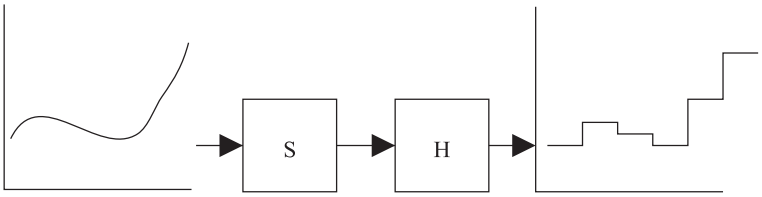


Figure 3-16: Zero order hold followed by a sampler.

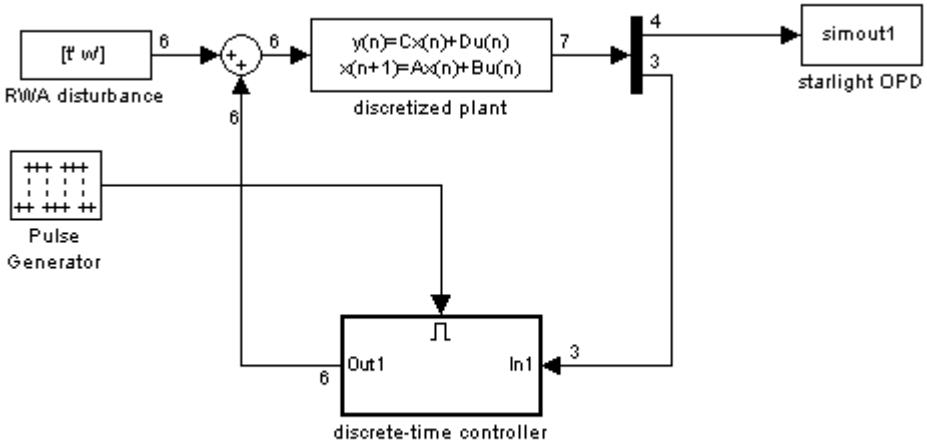


Figure 3-17: SIMULINK block diagram of a sampled-data control system.

the open-loop simulator `new_lsim.m`. Point-to-point error and relative RMS error plots are given in Figure 3-19. The result in Figure 3-19 clearly shows that the accuracy of

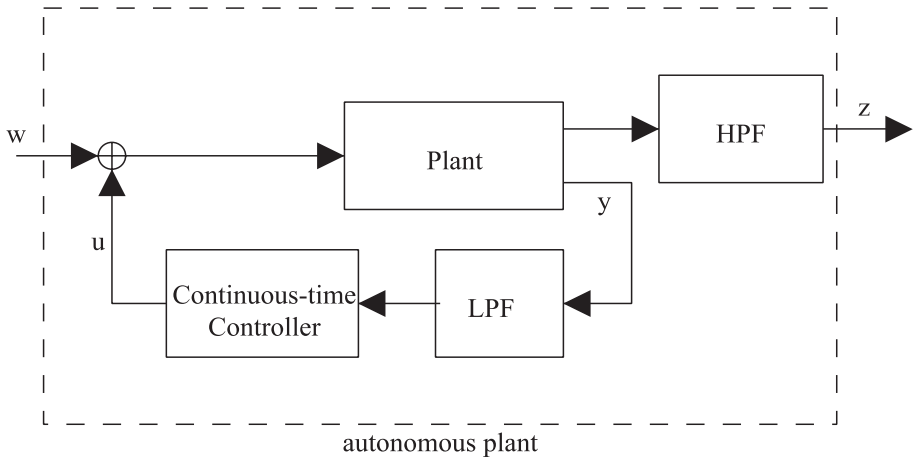


Figure 3-18: Closed loop system simulated by `new_lsim.m`.

continuous-time approximation improves with the increase of controller sampling rate.

There exist some other discretization schemes like bilinear transformation, response invariance, first order hold and so on. The same argument with the error is valid for these approximation since the holders are never ideal.

Now let's relax the assumption that the input to the discrete-time controller (i.e. measurement) is bandlimited. The input usually contains some high frequency components because RWA disturbances are generally broadband. However, the measurement is usually considered bandlimited because of the existence of the anti-aliasing filter preceding the controller. For the purpose of study, let's remove the controller and see what happens. This time, the external input is a sinusoidal signal (1Hz) and everything else is the same as before. The results for rigid body mode attitude angles are shown in Figure 3-20. It is clear that with anti-aliasing filter preceding the discrete-time control, the point-to-point error for the case in which the controller sampling rate is 1Hz is reduced significantly. It should be pointed out, though, that the difference between the point-to-point error where controller sampling rates are high is not due to aliasing. As a comprehensive testing, the point-to-point error for the performance output (starlight OPD) should be given as well, see Figure 3-21. It is interesting to note that the difference between the point-to-point errors for the low controller sampling rate case in Figure 3-21 is not obvious since starlight OPD's are less sensitive to the attitude control system, which aims at stabilizing the rigid body modes. Also, starlight OPD's are filtered by a high-pass filter, which simulates the effect of optical control, before they are measured, so the low frequency response due to ACS control

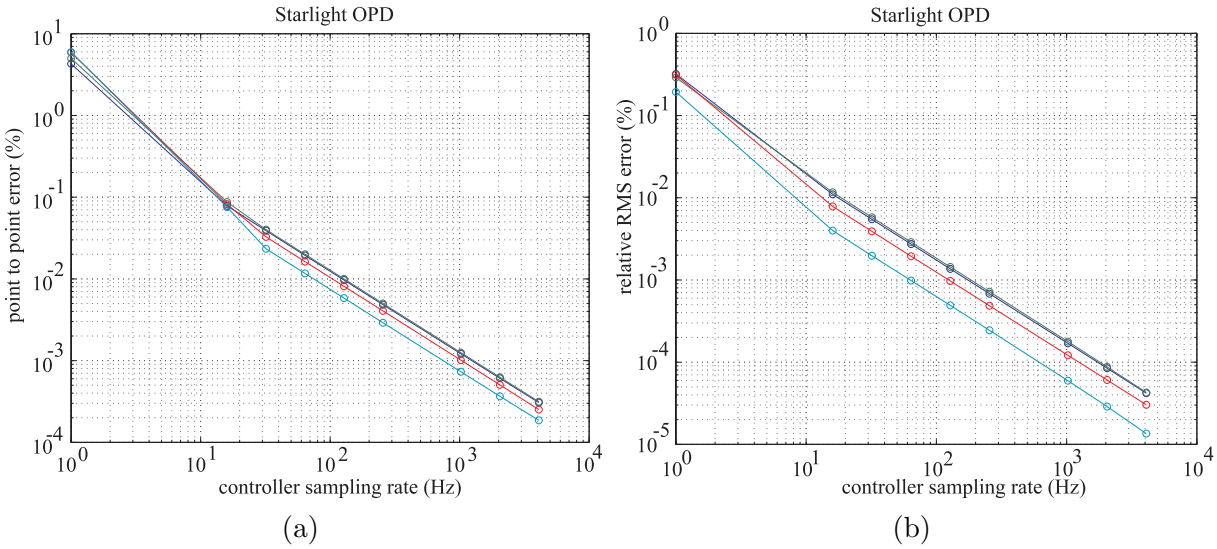


Figure 3-19: (a) Logarithmic plot of relative point-to-point error for starlight OPDs. (b) Logarithmic plot of relative RMS error for starlight OPDs.

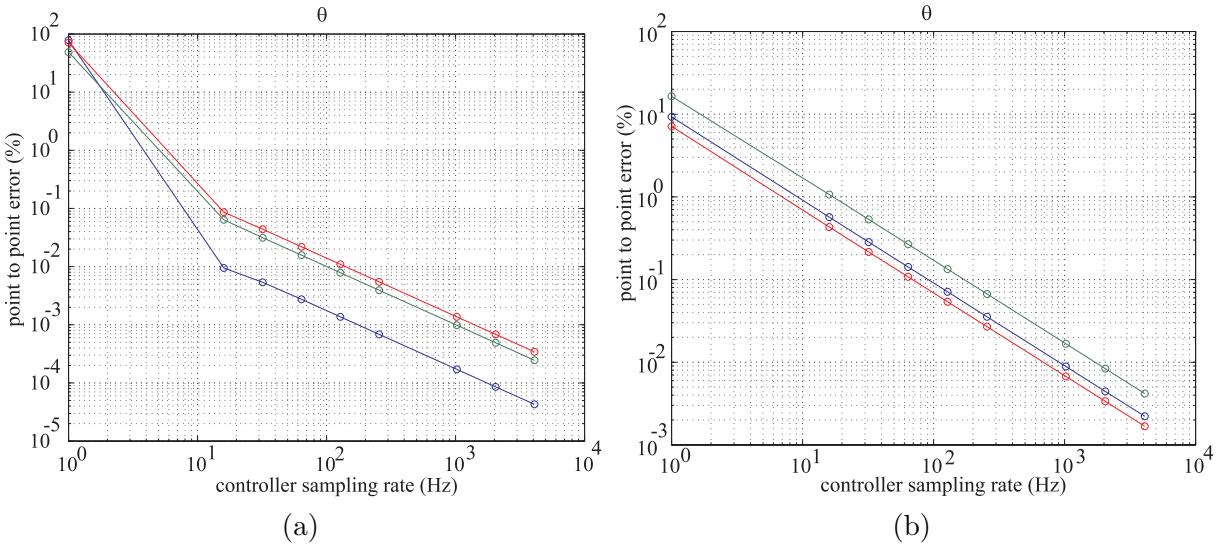


Figure 3-20: (a) Logarithmic plot of relative point-to-point error of rotational rigid body mode angles. Without pre-filtering.(b) Logarithmic plot of relative point-to-point error of rotational rigid body mode angles. With pre-filtering.

effort is not reflected as clearly as the case in rigid body mode attitude measurements.

In conclusion, the method of continuous-time approximation of discrete-time controller can be applied if the controller sampling rate is high enough (i.e. the discrete-time controller does not suffer from aliasing effects, which is the combined effect due to input and system. For example, for the example with the RWA disturbance discussed here, 10 Hz is good

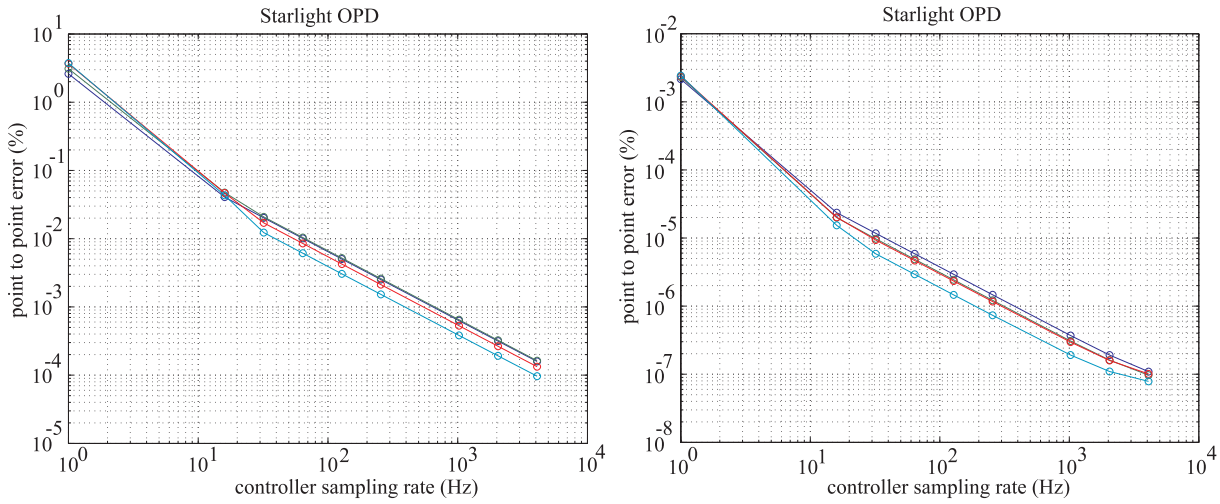


Figure 3-21: (a) Logarithmic plot of relative point-to-point error of starlight OPDs. Without pre-filtering. (b) Logarithmic plot of relative point-to-point error of starlight OPDs. With pre-filtering.

enough) and the accuracy is improved with the analog anti-aliasing filter, which is usually implemented for reasons other than simulation.

3.3.3 Conversion to LTI Systems by Lifting

In this subsection the other approach to dealing with the multiple sampling rates problem is discussed. This procedure is lifting as is studied in [33]. The basis of this procedure begins with the lifting of discrete-time signals.

Suppose there is a signal $v = \{v(0), v(1), v(2), \ldots\}$, where $v(i) \in \mathbb{R}^p \quad \forall\ i \in \mathbb{N}$ and p is the dimension of the signal. v is sampled with sampling period T. Then the lifted signal is

$$\underline{v} = \left\{ \begin{bmatrix} v(0) \\ v(1) \\ \vdots \\ v(n-1) \end{bmatrix}, \begin{bmatrix} v(n) \\ v(n+1) \\ \vdots \\ v(2n-1) \end{bmatrix}, \ldots \right\}. \tag{3.18}$$

n in (3.18) is the factor of lifting and must be an integer. The dimension of the lifted signal $\underline{v}$ is np and the sampling period is $\frac{T}{n}$. Define the lifting operator L and it is graphically shown in Figure 3-22. In Figure 3-22, a slow-rate signal $\underline{v}$ is shown with slow-frequency dots and a fast-rate signal v is shown with fast-frequency dots. The inverse of lifting, denoted as L^{-1}, is the inverse assignment of (3.18).

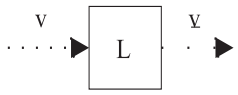


Figure 3-22: Schematic of lifting operation.

The next step is to lift a system. Let's prove an important theorem regarding lifting first. Given an LTI system G = ss(A,B,C,D) and the lifted system $\underline{G}$ in Figure 3-23 can be found by the following theorem.

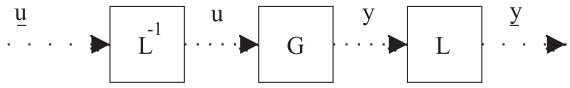


Figure 3-23: The lifted system.

Theorem 2 *The lifted system $\underline{G}$ of an LTI G is also LTI, and it has the following form*

$$\underline{G} = \left[\begin{array}{c|ccc|c} A^n & A^{n-1}B & A^{n-2} & \cdots & B \\ \hline C & D & 0 & \cdots & 0 \\ CA & CB & D & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{n-1} & CA^{n-2}B & CA^{n-3}B & \cdots & D \end{array}\right].$$

Proof We shall only prove the result when n=2. The general case follows readily. The system matrix [33], [13] of $\underline{G}$ is

$$\begin{aligned} [G] &= [L][G][L^{-1}] \\ &= [L]\begin{bmatrix} D & 0 & 0 & 0 & \cdots \\ CB & D & 0 & 0 & \cdots \\ CAB & CB & D & 0 & \cdots \\ CA^2B & CAB & CB & D & \cdots \\ \vdots & \vdots & \vdots & 0 & \ddots \end{bmatrix}[L^{-1}] \\ &= \left[\begin{array}{cc|cc|c} D & 0 & 0 & 0 & \cdots \\ CB & D & 0 & 0 & \cdots \\ \hline CAB & CB & D & 0 & \cdots \\ CA^2B & CAB & CB & D & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right] \end{aligned}$$

From the upper left block in the last equality we can identify $\underline{D}$. From "row" 2 "column" 1 we can identify $\underline{B}$ and $\underline{C}$. $\underline{A}$ is found by looking at "row" 3 "column" 1, which is not shown. Q.E.D.

This theorem will be useful later in the development.

A simple example is given here to illustrate how lifting works. Here a system ($A = 2, B = 1, C = 1, D = 0$) is simulated with the input vector $u =$ [1 2 3 4 5 6]. The lifted input is computed via (3.18) and the lifted system is calculated by the theorem just discussed. The sampling time is unity.

```
>> u = 1:6;                                                  % define input vector
>> u_lifted = zeros(2,3); u_lifted(:) = u(:);     % define lifted input vector
>> system_original = ss(2,1,1,0,1);               % define the original system
>> system_lifted = ss(4,[ 2 1 ],[ 1 2 ]',[ 0 0 ; 1 0 ],2);     % lifted system
>> y_original = lsim(system_original,u);        % simulate the original system
>> y_lifted = lsim(system_lifted,u_lifted);       % simulate the lifted system
>> y_original'
ans =
     0     1     4    11    26    57
>> y_lifted'
ans =
     0     4    26
     1    11    57
```

Now it is clear that the state propagates exactly the same for the original and lifted system. Moreover, exact output information is obtained from the two procedures.

With the tools at hand, now the problem in Figure 3-7 can be solved by applying the lifting and inverse lifting operators. Then the multirate system is as Figure 3-24. If only slow rate signals are concerned, then the system in Figure 3-24 can be further simplified into

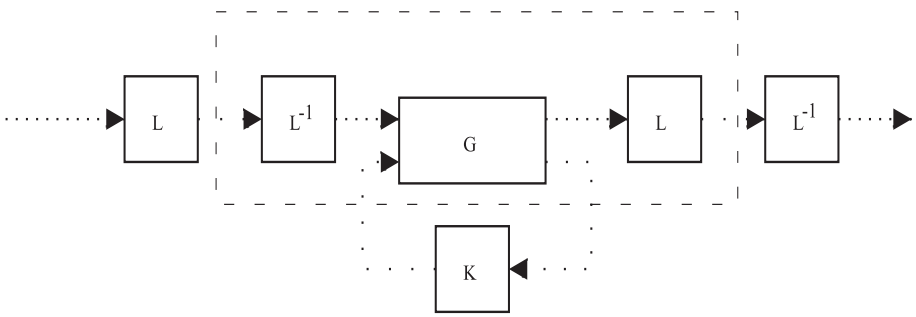


Figure 3-24: Two-rate system with lifting and inverse lifting.

a single-rate system like Figure 3-25. Analytically, if the original plant G has the following state space form

$$G = \left[\begin{array}{c|cc} A & B_w & B_u \\ \hline C_z & D_{zw} & D_{zu} \\ C_y & D_{yw} & D_{yu} \end{array}\right],$$

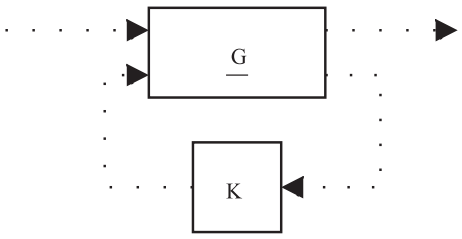


Figure 3-25: Single-rate system as a result of lifting.

where B_w and B_u are the input matrices related to external disturbance and control inputs respectively. C_z and C_y are the sensitivity matrices related to performance and measurement output respectively. The meanings of the subscripts of D-matrices can be deduced by the above rules. In order to obtain the ABCD matrices of the single-rate system in figure 3-25, two discretizations (this can be a problem for very large-order systems) are needed.

$$[A_d, B_{ud}] = c2d(A, B_u, T)$$
$$[A_f, [B_{wf}, B_{uf}]] = c2d(A, [B_w, B_u], T/n),$$

where T is the large sampling period and $n \in \mathbb{N}$ is the ratio between large sampling period and small sampling period and $c2d$ stands for continuous-time to discrete-time conversion. With these information and the theorem above, the state space representation of the lifted system in Figure 3-25 is given as

$$\underline{G} = \left[\begin{array}{c|cc} A_d & \underline{B}_w & B_{ud} \\ \hline \underline{C}_z & \underline{D}_{zw} & \underline{D}_{zu} \\ C_y & \underline{D}_{yw} & D_{yu} \end{array}\right] \tag{3.19}$$

where the matrices are

$$\underline{B}_w = \begin{bmatrix} A_f^{n-1}B_{wf} & A_f^{n-2}B_{wf} & \cdots & B_{wf} \end{bmatrix} \tag{3.20}$$

$$\underline{D}_{zw} = \begin{bmatrix} D_{zw} & 0 & \cdots & 0 \\ C_zB_{wf} & D_{zw} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C_zA_f^{n-2}B_{wf} & C_zA_f^{n-3}B_{wf} & \cdots & D_{zw} \end{bmatrix} \tag{3.21}$$

$$\underline{D}_{yw} = \begin{bmatrix} D_{yw} & 0 & \cdots & 0 \end{bmatrix} \tag{3.22}$$

$$\underline{C}_z = \begin{bmatrix} C_z \\ C_z A_f \\ \vdots \\ C_z A_f^{n-1} \end{bmatrix} \tag{3.23}$$

$$\underline{D}_{zu} = \begin{bmatrix} D_{zu} \\ C_z B_{uf} + D_{zu} \\ \vdots \\ \sum_{i=0}^{n-2} C_z A_f^i B_{uf} + D_{zu} \end{bmatrix} \tag{3.24}$$

With this LTI system ready, the conventional LTI analysis tool can be applied to obtain the response of this system.

3.4 Simulation Results

In this section some examples will be given as applications of the methods discussed in this chapter. The first example concerns the problem of a hybrid system and the block diagram is shown in Figure 3-26. The plant contains 406 state variables and is MIMO with 6 inputs

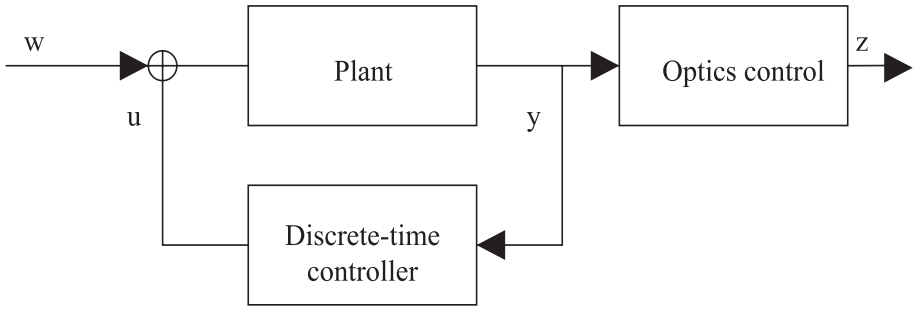


Figure 3-26: Block diagram of the example hybrid system. The closed loop system contains a discrete-time ACS and a highpass optics filter. The ACS is designed by LQG approach and the optics controller is a high-pass filter with corner frequency at 100 Hz.

(RWA Fx, Fy, Fz, Mx, My and Mz) and 4 outputs (starlight OPD 1, 2, 3, 4), the optics control contains 8 state variables and the discrete-time controller has 18 state variables respectively. The fast sampling rate is 4096 Hz and the controller sampling rate is 32 Hz. The input is again the Magellan reaction wheel assembly disturbance (1×10^5 samples). Three methods are used to simulate this problem.

1. Direct application of the SIMULINK discrete state space block as shown in Figure

3-17.

2. Converting the multirate sampled-data system to a single rate system by lifting procedure. The resultant system is then simulated by the standard simulator `lsim.m`. The procedure is coded as `sd_sim.m`[14].

3. Converting the hybrid system to a continuous-time system by applying the continuous-time approximation to the discrete-time controller. The resultant system is then transformed to real modal canonical form and simulated by `new_lsim.m`.

Some of the input and output signals are shown in Figure 3-27 (The outputs obtained from the three methods described above are not distinguishable by inspection). The result

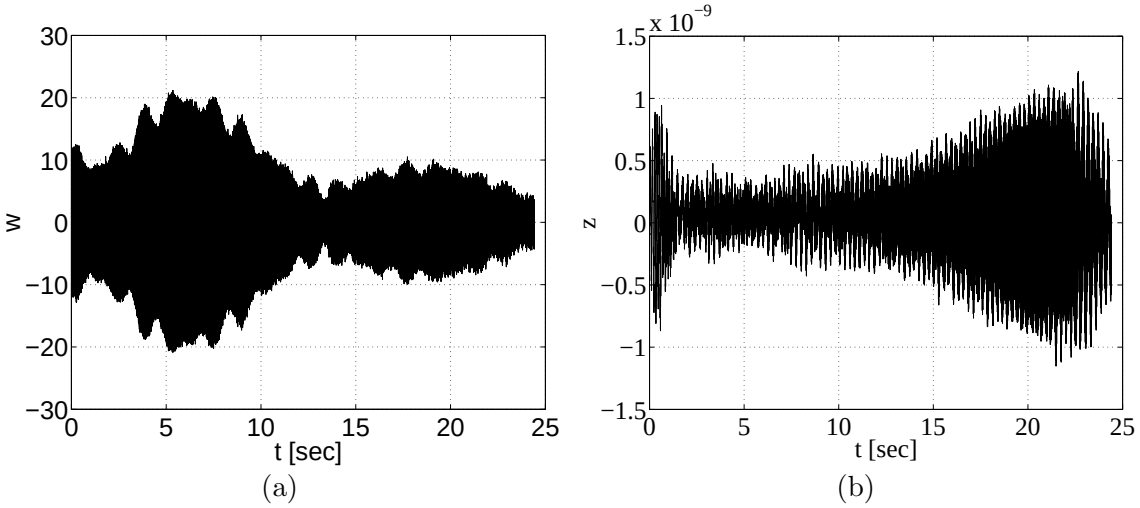


Figure 3-27: (a) Waveform of one of the input channel (RWA F_x). (b) Waveform of one of the output channel (Starlight OPD # 1).

obtained by SIMULINK simulation is chosen as the reference since this method does not require any approximation other than discretizing the plant. The accuracy of the results

Table 3.8: Point-to-point errors (%) of various methods.

Output channel	1	2	3	4
ϵ_{p2p}(%) of method				
`sdsim.m`	9.6045×10^{-7}	1.0125×10^{-6}	6.6674×10^{-7}	6.6456×10^{-7}
CT approx.	0.0152	0.0159	0.0106	0.0104

is listed in Table 3.8. It is interesting to note that the error level due to continuous-time

[14]This is homework problem 8.9 in [33].

approximation of the controller is consistent with the result described in Figure 3-19 in Subsection 3.3.2.

The conclusion of this example is that the continuous-time approximation is quite accurate and the lifting procedure provides highly accurate results. The simulation times, although not important in this relatively small problem, are given here to make the example complete. The computation times for direct application of SIMULINK and the continuous-time approximation followed by `new_lsim.m` are basically the same at about 8 seconds. The computation time for `sd_sim.m` is about 30 seconds. However, it should be pointed out that `sd_sim.m` uses a different discrete-time system solver, namely, `ltitr` and a 400-mode system is considered "large" for it.

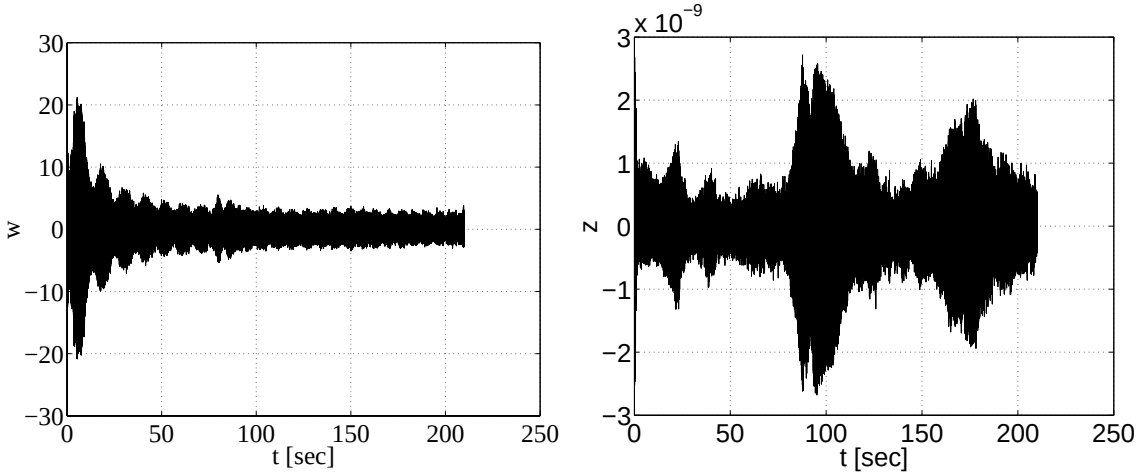
Another example is given here to illustrate the methods to handle the dynamics coupling introduced by feedback. The block diagram of the current problem is the same as Figure 3-26 except that the plant is of larger order (>2200 state variables). Also, the problem assumes that the plant has a block diagonal A-matrix with rigid body modes and flexible modes. The size of the system is as follows. There are 2180 state variables for the flexible mode subsystem (2172 state variables for the plant and 8 state variables for optics control), 14 state variables for rigid body mode subsystem (6 state variables for rigid body modes and 8 state variables for optics control). There are 6 RWA disturbance inputs and 4 optics outputs. The number of samples is 860160. The controller is designed by LQG approach and has 12 state variables and is considered continuous-time by the continuous-time approximation procedure. Three methods are again applied here.

1. Discretize the system by `c2d.m` and then directly apply SIMULINK as shown in Figure 3-17.

2. Formulate the closed loop system and apply `ss2mod71.m` to diagonalize it. The resultant modal system is then simulated by open loop simulator `new_lsim.m`.

3. Apply forced decoupling as described by the three step procedure in Figure 3-10. Both the rigid body mode subsystem and the flexible mode subsystem are simulated by the open loop simulator `new_lsim.m` and the responses are superposed. There is the flexibility that the closed loop rigid body mode subsystem can be simulated by other simulators, for example, `sd_sim.m`.

The waveforms of one of the input channels (RWA F_x) and output channels (starlight OPD #1) are shown in Figure 3-28. The CPU time and accuracy results are listed in



(a) (b)

Figure 3-28: (a) Waveform of one of the input channel (RWA F_x). (b) Waveform of one of the output channel (Starlight OPD # 1).

Table 3.9: CPU time and relative accuracy of various methods.

	SIMULINK	rediagonalization	forced decoupling
CPU time	$\approx$ 15 hours	$\approx$ 22 minutes	$\approx$ 7.5 minutes
ϵ_{p2p} (%) output ch1	0	7.2915×10^{-5}	3.9974×10^{-4}
ϵ_{p2p} (%) output ch2	0	1.4826×10^{-5}	7.8026×10^{-5}
ϵ_{p2p} (%) output ch3	0	1.5122×10^{-4}	3.4077×10^{-4}
ϵ_{p2p} (%) output ch4	0	3.3275×10^{-5}	1.3962×10^{-4}

Table 3.9. The structure of the computation time deserves some more elaboration since it might reveal possible space for improvement. Within the 15 hours for direct SIMULINK application, 445 seconds were used to discretize the system and the rest of the time was spent on the time integration chunk. Within the 22 minutes ($\approx$ 1320 seconds) for the rediagonalization procedure, 1022 seconds were spent on the rediagonalization procedure (over 90% of the effort was spent on computing the eigenvalues and eigenvectors) and the rest of time ($\approx$ 300 seconds) was spent on discretization and simulation. Within the 7.5 minutes ($\approx$ 450 seconds), about 90 seconds were used to simulated the closed loop rigid body modes subsystem, 45 seconds were used to cascade the flexible mode subsystem with optics control and the rest 300 seconds were used for the simulation of the large-order system. A few points can be concluded from the above results.

- The structure of the A-matrix of a dynamical system is crucial to the speed of simulation. The simulation time for the "same"[15] system can be reduced from 15 hours to 300 seconds (a factor 180 time savings) due to the (block) diagonal structure. Of course, there is an overhead for the advantage, though. The diagonalization takes about 17 minutes. Also, the eigenvectors obtained numerically can sometimes be highly ill-conditioned, which is a problem not considered by this thesis.

- The number of channels of the output is also an important factor in deciding the speed of the simulation. For example, the simulation time for the 30-mode rigid body mode subsystem is 90 seconds (about 1/3 of the computation time of simulating the 2000-mode large-order system) only because 6 more outputs (for step 2 in Figure 3-10) are required to be stored. However, it is the author's belief that this is a major room for improvement since it is more like a problem of memory management rather than a computation problem. Moreover, the problem is in a sense peculiar to the current example in that the number of samples involved is unusually large (8601600 entries or about 65Mb of memory).

- The problem of rediagonalization due to optical control (see Figure 3-26) can be relieved if the original plant is block diagonal (e.g. the eigenvalues and mode shapes are known) and the optical controller does not have the same eigenvalues with the plant. The reasons are as follows. The A-matrix of the cascade of two systems (see Figure 3-29) is not in general block diagonal, even if they are individually. Figure 3-30 shows the structure of the resultant A-matrix graphically and the A-matrix can

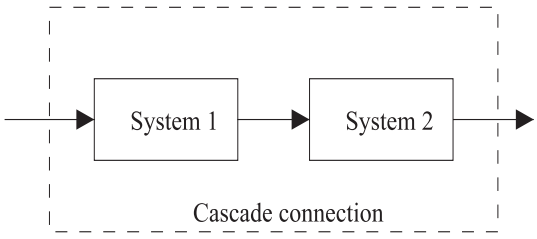


Figure 3-29: Block diagram of the cascade connection of two systems.

[15]The input-output relationship of a dynamical system is invariant under similarity transform. This observation is also manifested in the fact that the transfer function matrix is not changed by similarity transform.

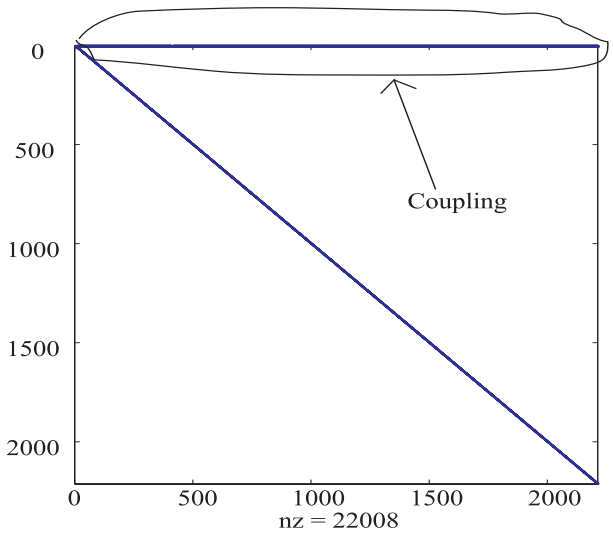


Figure 3-30: Sparsity plot of the A-matrix of a series connection of two block diagonal systems.

be expressed as

$$\begin{bmatrix} A_2 & B_2C_1 \\ 0 & A_1 \end{bmatrix}, \tag{3.25}$$

where A_1 is the A-matrix of the first system, A_2 is the A-matrix of the second system, B_2 is the B-matrix of the second system and C_1 is the C-matrix of the first system. However, the following Lemma [12] allows a similarity transform such that the transformed system is block diagonal.

Lemma 1 *Let $T \in \mathbb{C}^{n\times n}$ be partitioned as follows:*

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

where $T_{11} \in \mathbb{C}^{p\times p}$ and $T_{22} \in \mathbb{C}^{q\times q}$. Define the linear transformation $\phi : \mathbb{C}^{p\times q} \to \mathbb{C}^{p\times q}$ by

$$\phi(X) = T_{11}X - XT_{22}, \tag{3.26}$$

where $X \in \mathbb{C}^{p\times q}$. Then ϕ is nonsingular if and only if $\lambda(T_{11}) \cap \lambda(T_{22}) = \emptyset$, where $\lambda(T_{11})$ and $\lambda(T_{22})$ are sets of eigenvalues of T_{11} and T_{22} respectively. If ϕ is nonsingular and Y is defined by

$$Y = \begin{bmatrix} I_p & Z \\ 0 & I_q \end{bmatrix} \qquad \phi(Z) = -T_{12},$$

then $Y^{-1}TY = diag(T_{11}, T_{22})$.

The proof of this lemma will not discussed here and the interested readers should consult [12] for the proof. The implication of the lemma is that the upper block triangular A-matrix resulted from cascading two block diagonal systems can be converted into a block diagonal matrix by a similarity transform, where the transform matrix can be obtained from (3.26), a problem which can be solved efficiently because of the block diagonal A-matrices of the cascading systems. The implementation of this procedure is coded in `d_cascade.m`.

- Different schemes of discretization introduces some discrepancy between the results and it is reflected in Table 3.9 that the relative point-to-point error is not zero to machine precision (2.2204×10^{-16} in MATLAB). However, this "error" should be considered small compared with the gain in efficiency.

- Forced decoupling shows its value here in that the computation time is further reduced and the level of accuracy is maintained. The result is anticipated by the error estimation but the motivation of this approximation should come from the physical understanding of the problem and the engineering judgement that the measurement is dominated by the response of the rigid body modes.

3.5 Chapter Summary

In this chapter, the problem of closed loop system simulation is first defined and it is followed by the introduction of the challenges thus induced. The two simulation problems are identified as dynamics coupling and coexistence of continuous-time and discrete-time states, which can be referred to as a hybrid system problem in the context of digital control system design. Then, for each problem, two solutions (approximations or manipulations) are proposed. For the problem of dynamics coupling, a rediagonalization procedure can be applied to convert the closed loop system back to an open loop system, which can be simulated with open loop simulators like `new_lsim.m` described in Chapter 2. The other solution (forced decoupling) to dynamics coupling is more ambitious but risky in that some of the couplings are considered weak and disregarded. This approximation results in a sequential simulation procedure which can preserve the original block diagonal structure of the open loop system. For the problem of hybrid system, the first suggestion is to approximate the discrete-time controller with a continuous-time one in order to make the overall system tractable. This

approximation is shown to be good if the sampling rate of the controller is high enough. The other suggestion for hybrid system simulation is the conversion made available by the lifting procedure, which exploits the mapping characteristic of the solution to ODE IVP's, which is the state transition formula in the context of LTI system simulation. After the principles of the proposed solutions are studied, error analysis and estimation procedures are suggested, especially for heuristic methods like forced decoupling and continuous-time approximation. Finally, two examples are given to illustrate the methods to tackle the two problems discussed throughout this chapter to show their values.

Graphically, the methods suggested in this chapter can be summarized in Figure 3-31.

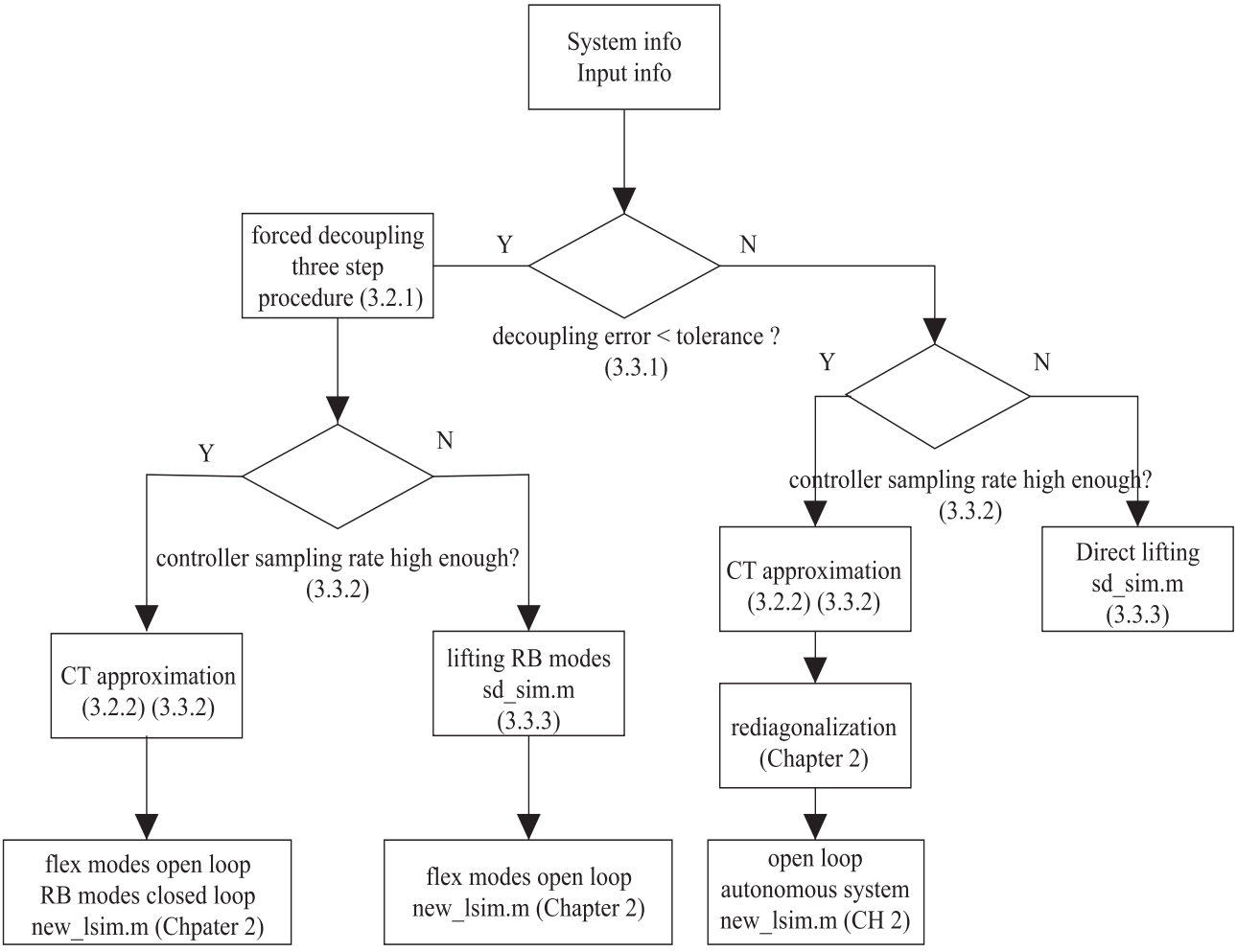


Figure 3-31: The decision tree of the methods described in Chapter 3.

In the next chapter, the methods and tools developed previously will be applied to the analysis of the model of the Space Interferometry Mission (SIM).

Chapter 4

Application

As an application to real world applications, the simulator (`new_lsim.m`) developed in Chapter 2 and the simulation methods suggested in Chapter 3 will be applied to the analysis of the Space Interferometry Mission (SIM v2.2).

4.1 SIM model description

The Space Interferometry Mission (SIM) is one of the projects of NASA's ORIGINS program which aims at gaining more understanding of our age old questions about the universe: How big is the universe? How old is it? SIM will be able to measure the positions of stars several hundred times more accurately than any other previous programs. Also, SIM will pioneer an optical interferometry technique called nulling, which will block out the star's light so that the faint orbiting planets can be measured. In order to achieve the above scientific objectives, SIM will be required to meet very stringent performances. For example, the optical path length should not deviate 10nm and 1nm from its nominal value for astrometry and nulling modes respectively. Laser metrology should not have an error greater than 50pm[1].

The SIM flight system consists of the interferometry instrument system and the spacecraft system. The spacecraft system provides engineering functions for flight operations such as structure, attitude control and communication. The instrument system includes optics, sensors, actuators and computers needed to make scientific observations.

SIM can be modelled as an LTI system with controls such as ACS and optical control. Figure 4-1 shows the appended LTI system model of SIM. The opto-structural plant in

[1]See http://sim.jpl.nasa.gov/technology/index.html for more detail.

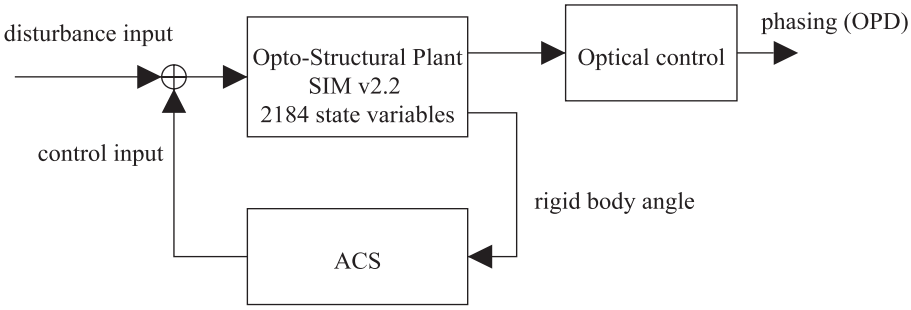


Figure 4-1: Appended LTI system dynamics of SIM. The opto-structural plant is obtained by combining the optics and the finite element models (FEM). The attitude control system (ACS) is necessary to stabilize the open-loop unstable rigid body modes. The optical control system is added to improve optical performance.

Figure 4-1 can be represented by the standard LTI system,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{4.1}$$

where $x(t)$ is the state vector of the plant, $u(t)$ is the input (disturbance and control) vector, $y(t)$ is the output vector, A determines the linear dynamics of the structural and optical modes. B maps the input to the dynamics of the opto-structural plant. C is the sensitivity matrix of the optical performance with respect to the dynamics of the plant, and D, if not zero, corresponds to the feedthrough from input to optical output. These matrices are obtained from relevant programs, such as `imos` developed by NASA [18]. The attitude control system (ACS) is necessary since the plant contains rigid body modes (rotational only in spacecraft design) which are open-loop unstable. The input to the ACS is the attitude angles[2] of the rigid body modes and the output is control input that stabilizes them. The ACS can be designed by classical methods like PID or lead-lag. Alternatively, it can be designed with MIMO controller synthesis tools such as LQG.

4.2 Performance Analysis with Optical Controller

In this section, the effect of the optical controller (in Figure 4-1) on some of the performance outputs will be studied. Before the time chunk begins, it is necessary to specify the

[2]A more realistic model will be the following: The angular velocities, instead of the attitude angles, are measured by the gyros, then the measured angular velocities are integrated to obtain the estimate of the angles, which are corrected periodically by measurement from star tracker. It is natural that various sensor noises can be added to increase the model fidelity.

requirements on the performance outputs, which are summarized in Table 4.1[3] [29]. With

Table 4.1: SIM opto-mechanical performance requirements.

performance z_i	units	requirement
Starlight OPD #1	nm	10 (RMS)
Internal Metrology OPD #1	nm	20 (RMS)
Starlight WFT #1	asec	0.210 (RSS)

the requirements in mind, the goal of the optical controller is to reduce the effect of the disturbances (RWA disturbance or other noise sources) so that the performance outputs are always within the acceptable operation range. The optical controller here is modelled as a high-pass filter and the transfer function of a single channel is

$$K_o(s) = \frac{s^2}{s^2 + 2\zeta_o\omega_o s + \omega_o^2},$$

where ζ_o is the damping ratio of the controller and is set to 0.707 and ω_o is the corner frequency, which is treated as a design parameter.

The settings of the simulation is in Figure 4-1, where the corner frequency of the optical controller, $f_o = \frac{\omega_o}{2\pi}$ is varied from 0 Hz (no control) to 1000 Hz (cheap control). The plant is the stabilized SIM model v2.2 with about 2200 state variables. The 6 inputs to the closed loop plant are RWA F_x, F_y, F_z, M_x, M_y and M_z. The 3 outputs are filtered by the optical controller and they are starlight OPD #1, internal metrology OPD #1 and starlight WFT #1. The input data is again the Magellan reaction wheel assembly disturbance (4096×210 samples). Since the emphasis here is the optical control, the ACS controller can be approximated by a continuous-time one, under the assumption that the sampling rate is high enough. The simulation results are summarized in Figure 4-2. See also Figure 4-3 for the waveforms of some characteristic cases. It can be concluded from Figure 4-2 that the optical controller is important to the performance of SIM and the corner frequency should be higher than 10 Hz in order to achieve the performance (Of course, this by no means guarantees that a 10 Hz controller corner frequency is all that is needed because the analysis here is simplified[4]). The reason why the disturbance output of internal metrology OPD #1 did not increase with the decrease of controller corner frequency is that the performance is not sensitive to the low frequency modes, especially rigid body modes. (See Figure 4-5).

[3]RSS stands for root-sum-square, which consists of RMS's of, say in 2D, x direction and y direction.
[4]One important omission is the low-frequency noise floor of a realistic controller, see Figure 4-4.

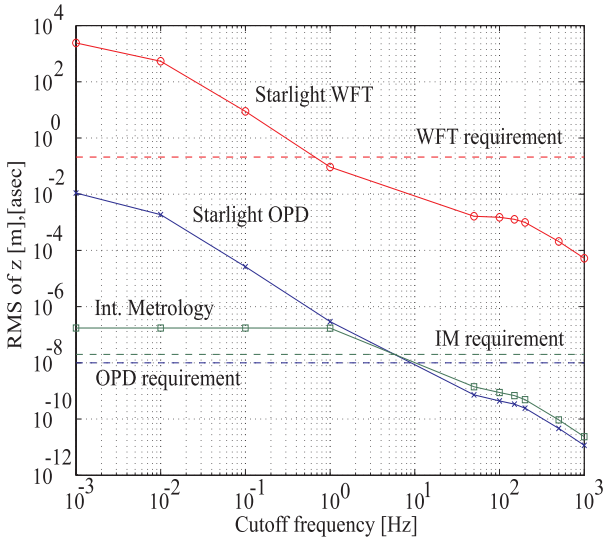
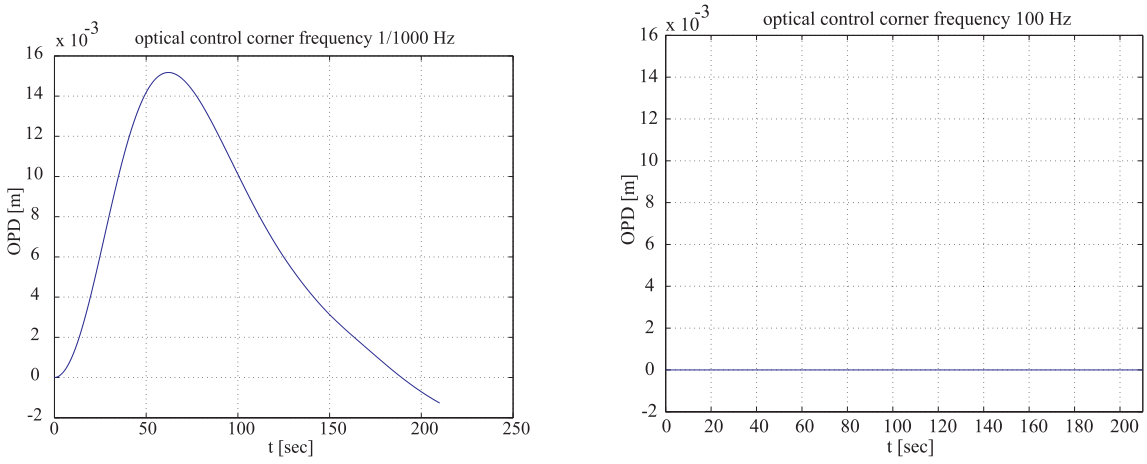


Figure 4-2: RMS and RSS of the performance outputs. Circle: Starlight OPD #1. Asterisk: Internal Metrology OPD #1. Square: Starlight WFT #1.

Figure 4-2 also implies the direction of performance improvement that one might want to look at: If the modal gain for the low frequency modes are reduced, then the performance of starlight OPD and starlight WFT can be improved significantly. Although from simulation it is implied that the performance increases with the increase of the optical controller corner frequency, the corner frequency should not be too high since it will impose a too difficult job for the optical control engineers and the cost of control will become too high. Finally, it should be pointed that the conclusion is based on the assumption that the disturbance input is due to the Magellan reaction wheel assembly, which is considered as "noisy". The results can be different when the disturbance model is different. For example, in [29] the disturbance source was Hubble Space Telescope (HST) RWA disturbance, which was very "quiet". The conclusion drawn from the paper was that the open (optical) loop and closed (optical) loop systems did not differ significantly.

4.3 Transient Response Analysis

In this section, the step response of the closed loop system will be analyzed. Because of the pole locations of the rigid body modes, the attitude angles of the closed loop system are guaranteed to have zero steady state error due to step input if unity feedback is employed as in Figure 4-6. The steady state accuracy of the optical performances (e.g. starlight



(a) (b)

Figure 4-3: Waveform for 210 seconds of starlight OPD # 1. (a) Optical control $f_o = \frac{1}{1000}$ Hz. (b) Optical control $f_o = 100$ Hz.

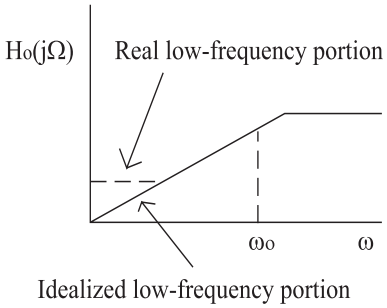
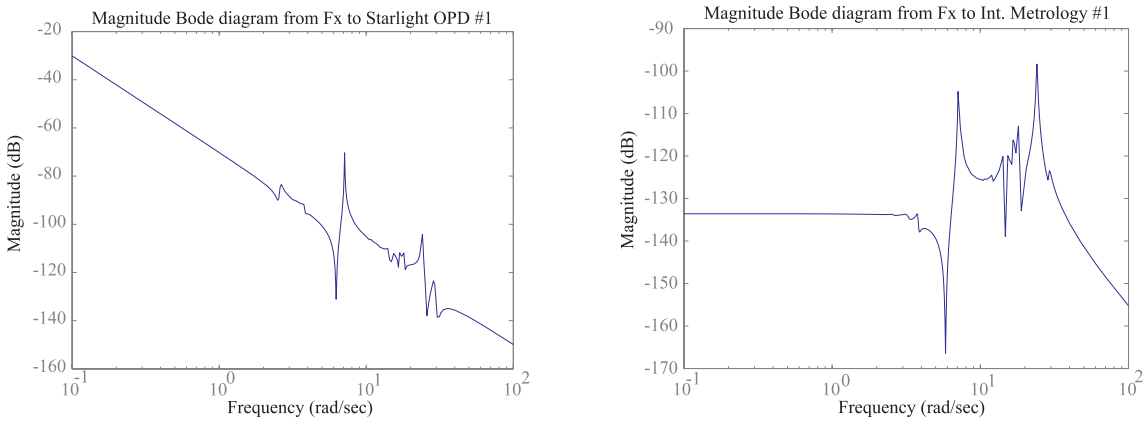


Figure 4-4: Magnitude transfer function of an idealized and a more realistic model of the optical controller.

OPD), however, are not guaranteed since they are neither sensed by the ACS sensor nor considered in the controller design. Fortunately, if the optical controller (high-pass filter) is implemented, steady state accuracy of the optical performances[5] will be achieved since the transfer function from the reference to the filtered output will have zero DC gain. This fact also manifests the need for an optical controller. The time responses of the attitude angles and starlight OPD #1 are shown in Figure 4-7. It can be seen from Figure 4-7 that the steady state error for the attitude angles due to step reference is zero, as expected. The steady state error for starlight OPD #1 is also zeros because of the optical controller. The transient responses, however, are quite unsatisfactory (the overshoot is high (about 60%) and the settling time is long (about 150 seconds)). The reason for the unsatisfactory

[5]This step scenario is somewhat hypothetical since optics loop will not be closed during large angle slew maneuvers. Nevertheless it is instructive to analyze step response from a control point of view.



(a) (b)

Figure 4-5: Open loop Bode diagram of the SIM model for the low frequency range. (a) Transfer function from F_x to starlight OPD # 1. (b) Transfer function from F_x to internal metrology # 1.

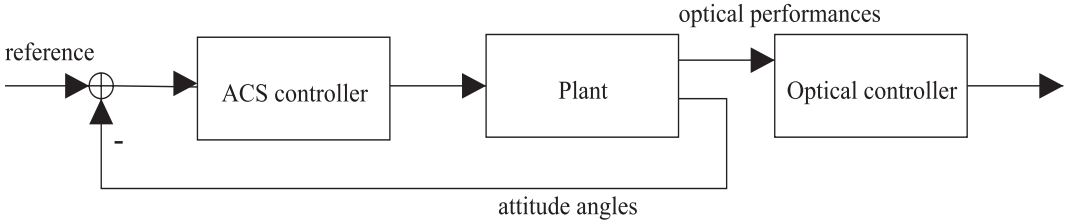


Figure 4-6: Block diagram of the SIM model with unity feedback gain.

transient response is twofold. One is due to control effort and the other is due to the characteristic of the plant. It is true that if more control effort is employed[6] , the settling time and percentage overshoot are expected to be smaller, as verified in Figure 4-8 (a). Here the settling time (for attitude angles) is about 30 seconds and the overshoot is about 40%. It should be pointed out that the performance of starlight OPD is not improved with more control (In fact, it gets worse.) because it is not measured by the ACS sensor, which means that the ACS has no authority to control it. While the settling time reduces with the increase of control effect, the overshoot is not improved significantly. The reason for this are nonminimum phase zeros, which makes the plant unamenable to control. The pole-zero diagram of the plant for controller design[7] is given in Figure 4-9. One of the ways to understand why nonminimum phase zeros are bad is the classical approach: On the

[6]In terms of LQR design, it means that the state is penalized with a higher cost.

[7]The plant used for controller design has only the rigid body modes and 25 flexible modes of the 2178 state variable SIM model v2.2. The other high frequency flexible modes are not considered by the ACS controller.

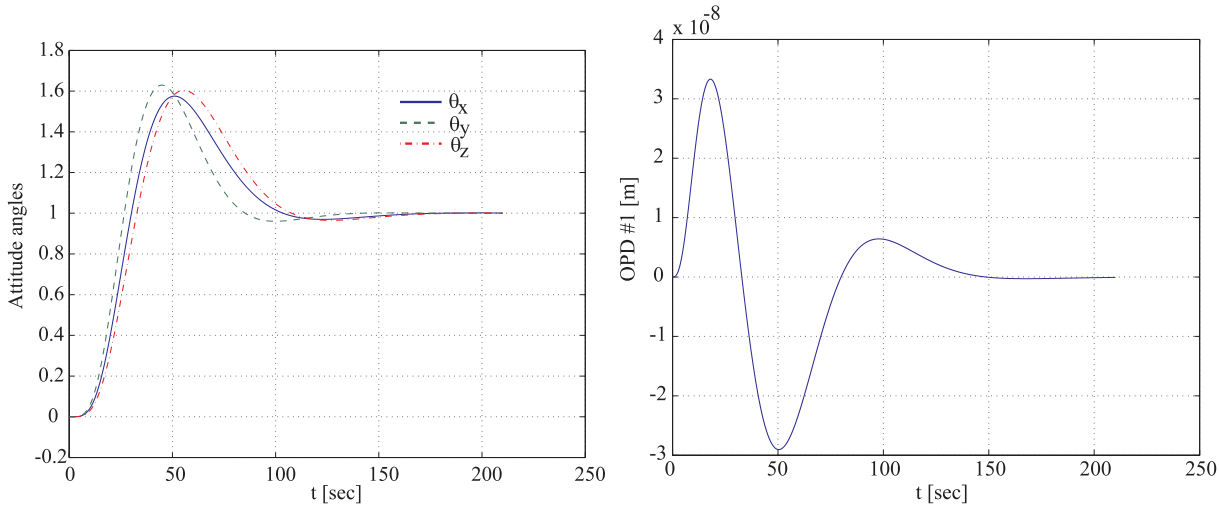


Figure 4-7: Transient response of the closed loop SIM model due to step reference. (a) Attitude angles. (b) Starlight OPD #1.

root locus plot, the closed loop poles evolve from open loop poles to open loop zeros as the controller gain is increased[8]. A stabilized plant has open loop zeros in the left half s-plane and the closed loop poles become more stable as the controller gain is increased. However, the nonminimum phase zeros impose limits on control effort in that some of the branches will be driven to right half s-plane if the controller gain is too high. This informal argument gives the underlying reason why the controller is not doing its job (improve transient performance) even if it is told to do so. The problems with control here point out a direction for the structural design, though. If the part of the structure responsible for the nonminimum phase zeros is identified and the NMZ's are eliminated or pulled further away from the imaginary axis, then the transient response is expected to be better. The above discussion was only from the point of view of control system design. In actual implementation, there are much more limitations that one might encounter. For example, the actual control effort is governed by RWA torque authority and S/C inertia. One final comment to the the control system structure is appropriate at this time. It was assumed throughout this section that unity feedback was employed. If the controller must be implemented in the feedback loop, rather than in cascade with the plant, then the reference input should be filtered by the same controller before being applied to the closed loop plant, as in Figure 4-10, which is

[8]This is true only for 180 degree root locus, i.e. negative feedback.

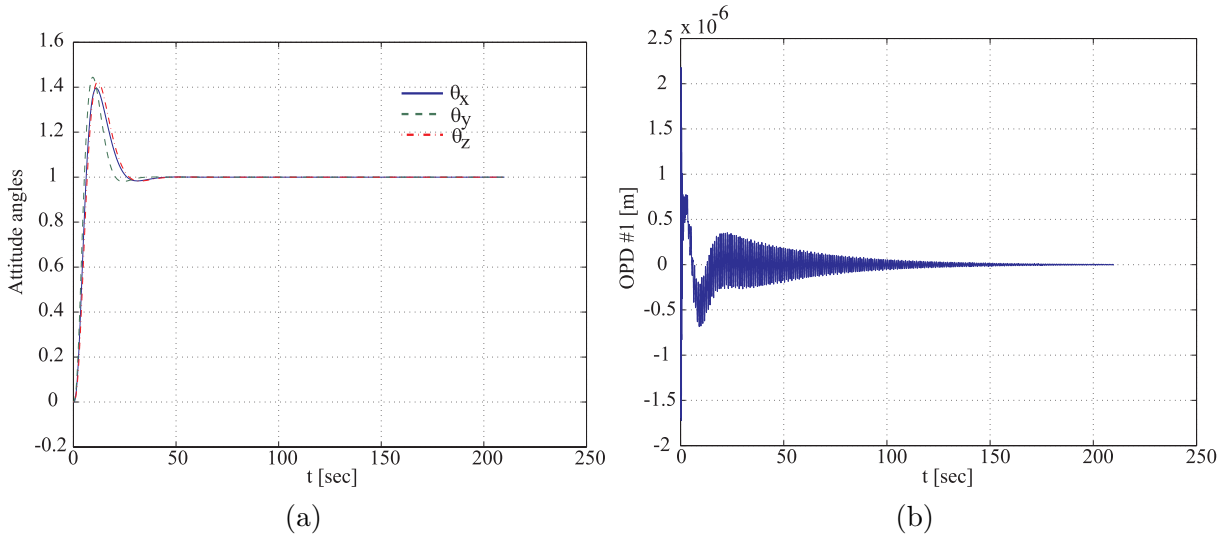


Figure 4-8: Transient response of the closed loop SIM model due to step reference. (a) Attitude angles. (b) Starlight OPD #1.

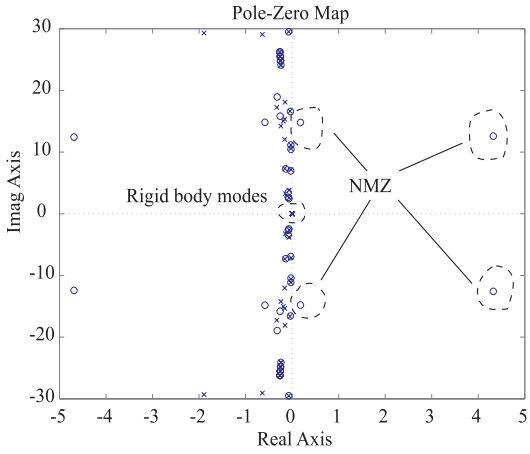


Figure 4-9: Pole-zero diagram of the open loop design model.

clearly equivalent to the closed loop system in Figure 4-6.

4.4 Sensitivity Analysis of Design Parameters

In this section some design parameters are chosen to see how the performances (e.g. starlight OPD, internal metrology and starlight WFT) are affected by perturbation of these parameters. The chosen parameters are

1. Bending moment inertia of a diagonal bar of the solar array (Bar 170400), p_{sp}.

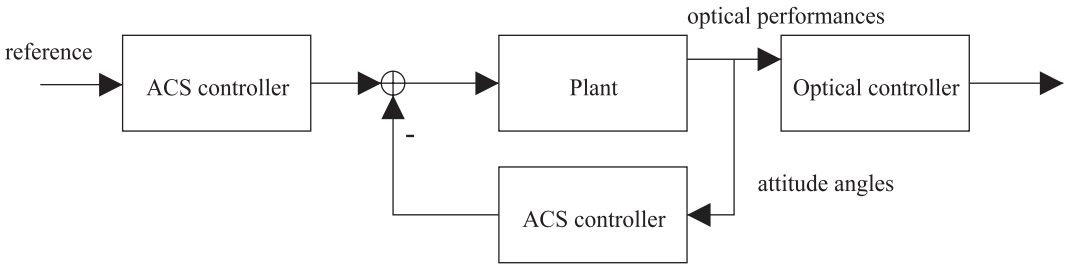


Figure 4-10: Equivalent closed loop system block diagram with the controller in the feedback loop.

2. Cutoff frequency of the optical controller, ω_o (or f_o).

3. Minimum optical mount frequency in PSS (precision support structure), p_{pss}.

4. Frequency factor (stiffness) of reaction wheel assembly isolator in backpack, p_{rwaiso}.

5. Backpack isolator stiffness (or backisoK) in backpack, p_{bp}.

Figure 4-11 shows the finite element model of SIM model v2.2 and indicates where the design parameters come from.

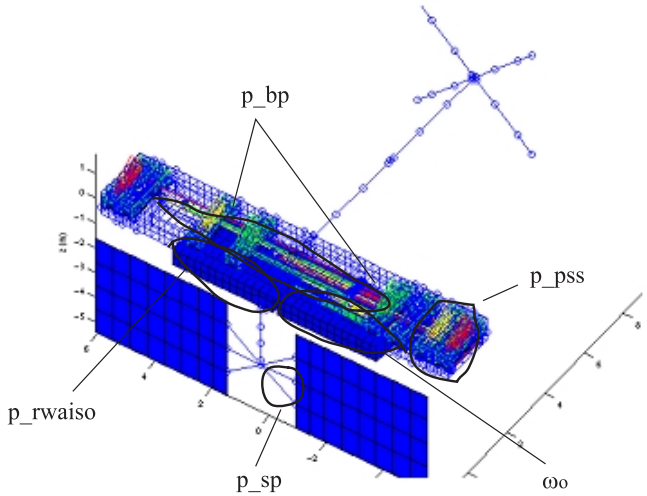


Figure 4-11: SIM finite element model and the locations of the design parameters.

In order to carry out a parameter study of the sensitivity of the system, it is necessary to compute the Jacobian matrix as follows:

$$\begin{bmatrix} \frac{\partial J_1}{\partial p_1} & \frac{\partial J_1}{\partial p_2} & \cdots & \frac{\partial J_1}{\partial p_n} \\ \frac{\partial J_2}{\partial p_1} & \ddots & & \\ \vdots & & & \\ \frac{\partial J_m}{\partial p_1} & & & \frac{\partial J_m}{\partial p_n} \end{bmatrix}, \tag{4.2}$$

where J_i is the i^{th} performance, p_j is the j^{th} parameter, m is the number of considered parameters and n is the number of the considered performances. It should be pointed out, though, that the "Jacobian matrix" mentioned here is not in the usual sense in that the Jacobian matrix of a system is formally defined as the partial derivative of the vector field with respect to the state variables of the system. In the rest of the section, Jacobian matrix should be considered as what is defined in (4.2). The Jacobian matrix can be obtained analytically (an elegant but more expensive way), or it can be obtained by finite difference method, which is less accurate but more efficient especially with the developed simulation capability. Therefore, finite difference approximation to the Jacobian matrix is chosen here. That is

$$\frac{\frac{\partial J}{\partial p}}{\frac{J_o}{p_o}} \cong \frac{\frac{\Delta J}{\Delta p}}{\frac{J_o}{p_o}} = \frac{\frac{J'-J_o}{p'-p_o}}{\frac{J_o}{p_o}}. \tag{4.3}$$

Here the derivative is normalized by the nominal values of the performance and parameter. The variation of parameter is set to be 5% and ΔJ is the difference of the performances in the original and perturbed settings. As before, simulations have been conducted and the settings are as follows. The inputs to the plant are again the three forces and three torques of the Magellan RWA running from 1500 to 2000 rpm and the nominal value of the optical controller corner frequency is 100 Hz. The results of the sensitivity analysis are summarized in Figures 4-12, 4-13 and 4-14. From the figures above one can conclude that the plant is most sensitive to the variations of the frequency factor of the reaction wheel assembly isolator (1 % increase in RWA isolator stiffness results in about 4 % increase in starlight OPD RMS.) and the minimum optical mount frequency, while it is least sensitive to the perturbation of the bending moment inertia of the solar array. The reason for the sensitivity due to the RWA isolator and minimum optical frequency is that these components directly affect the path between mechanical disturbance and optical performance metrics. It is also interesting to note that sensitivity of RMS WFT with respect to RWA isolator stiffness

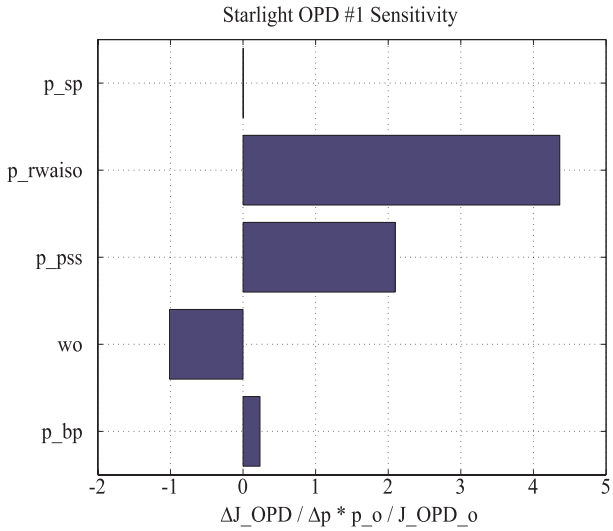


Figure 4-12: Sensitivity analysis of starlight OPD #1. See text for the meaning of the parameters.

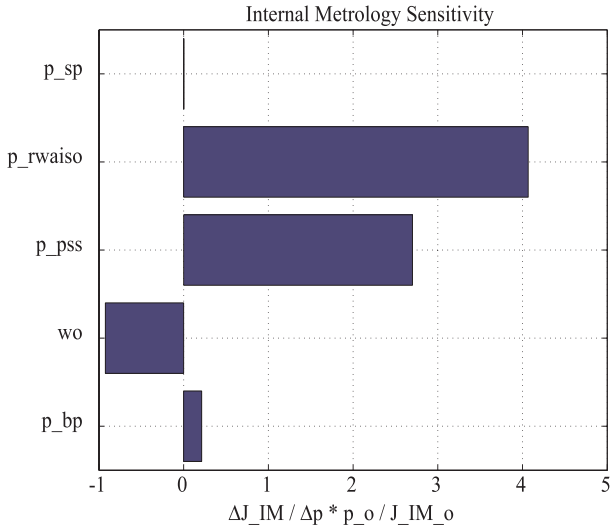


Figure 4-13: Sensitivity analysis of internal metrology #1. See text for the meaning of the parameters.

is opposite to that of OPD. This result is counterintuitive and further investigations are required to understand it. Finally, it should be noted that the analysis is by no means exhaustive and more analysis should reveal more interesting parameters and different finite difference schemes (e.g. central difference or backward difference) may result in somewhat different conclusions. There is more to be done. See recommendations (Chapter 5) for future work.

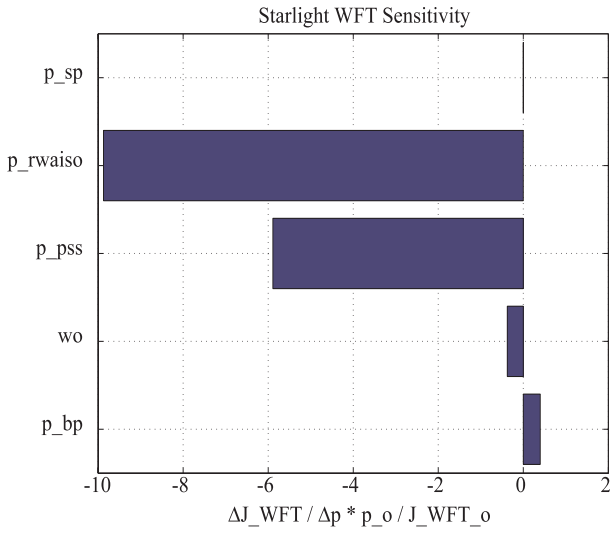


Figure 4-14: Sensitivity analysis of starlight WFT #1. See text for the meaning of the parameters.

4.5 Chapter Summary

In this chapter, various analyses have been conducted based on the model of SIM v2.2 with the tools and methods developed in Chapter 2 and 3. First, a parameter study of the corner frequency ω_o of the optical controller was conducted and it was found that as ω_o increases, the performances has smaller RMS values, which is reasonable because more control effort has been applied. Then, the transient responses of the SIM model due to unit step attitude angle reference were computed and it was found that the transient performance of the closed loop system was unsatisfactory because of the nonminimum phase zero in the plant. Finally, a sensitivity analysis was conducted and it was found that the frequency factor of the reaction wheel assembly isolator and the minimum optical mount frequency in the PSS were the most important ones considered so far.

Chapter 5

Conclusions and Recommendations

5.1 Summary

A new simulator, `new_lsim.m`, has been developed to solve large-order continuous-time LTI system simulation problems. In Chapter 1, the problems of simulation was first introduced at a system design level, and the discussion was followed by a summary of the technical problems that would be addressed in the subsequent chapters. Then the underlying assumptions of the simulator were discussed. After that, the philosophy of the new simulation scheme was briefly introduced. At the end of Chapter 1, an overview of the thesis was given with the thesis roadmap.

In Chapter 2, issues concerning the simulator, `new_lsim.m` were discussed in detail. In Section 2.1, the characteristics of the targeted problems were discussed briefly. Then, in Section 2.2 the flow chart of the simulator was given, followed by a brief discussion for each of the processes. After that, in Section 2.3 the basics for the manipulations of `new_lsim.m` were studied. In Section 2.4, detailed discussion of each of the implementations shown in Figure 2-4 was given. In that section, modal decomposition, subsystem planning, downsampling, discretization, simulation and interpolation details were addressed. From all the discussion above, it was found that the main advantages exploited by `new_lsim.m` were fast discretization and O(n_s) state transition allowed by the diagonal (block diagonal) structure of the A-matrix. After all the details on the implementations, simulation examples were given in Section 2.5 and the computation cost (FLOPS and CPU time) was discussed briefly. In the last section (Section 2.6) of this chapter the accuracy issues of `new_lsim.m` were addressed and an error estimation procedure was proposed.

In Chapter 3, the problem of closed loop system simulation was first defined. This was

followed by the introduction of the challenges induced by digital control of continuous-time system. The two simulation problems were identified as dynamics coupling and coexistence of continuous-time and discrete-time states, which can be referred to as a hybrid system problem in the context of digital control system design. Then, for each problem, two solutions (approximations or manipulations) were proposed. They were rediagonalization, forced decoupling, continuous-time approximation and lifting. It was found that rediagonalization was more general but required more computation, while forced decoupling was more restrictive, it could avoid the $O(n_s^3)$ eigenvalue problems. For the problem of multiple sampling rates, continuous-time approximation was proved to be a convenient and acceptable work around if the sampling rate of the controller was not too low, otherwise, the lifting procedure, which was not very well adapted to large-order problems, was required. After the studying principles of the proposed solutions, error analysis and estimation procedures were suggested, especially for heuristic methods like forced decoupling and continuous-time approximation. Finally, two examples were given to illustrate the methods to tackle the two problems discussed throughout this chapter to show their values.

In Chapter 4, the simulator developed in Chapter 2 and the methods developed in Chapter 3 were applied to the analysis of the Space Interferometry Mission (SIM).

5.2 Decision Tree

The decision tree of the various methods described in previous chapters is given as a graphical summary of the thesis.

5.3 Contributions

In this thesis the following contributions can be summarized:

1. Identified the bottleneck in the computation of time domain simulation for large-order LTI systems.

2. Proposed diagonalization solution to simulating large-order systems.

3. Studied the advantages and disadvantages resulting from diagaonalization.

4. Proposed, at a user level, a simulator that exploits the advantages resulted from the previous studies.

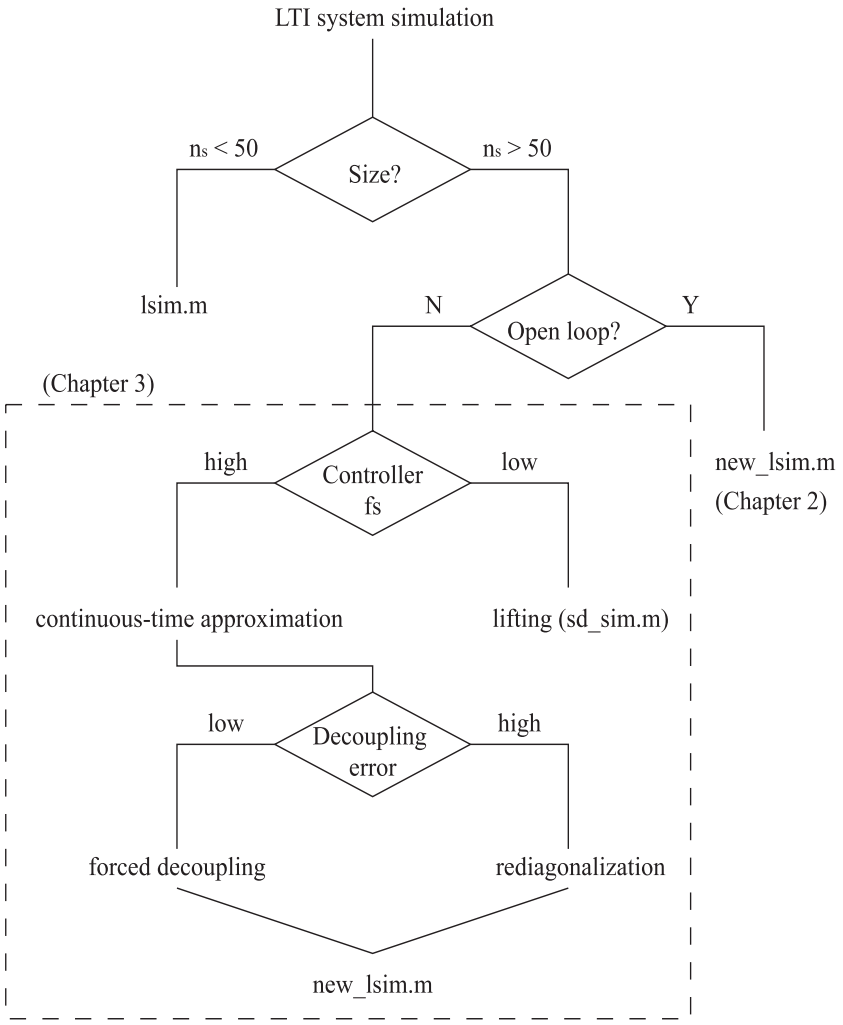


Figure 5-1: Decision tree of the LTI system simulation problem with `new_lsim.m` and other methods.

5. Suggested a computational cost and error estimation to serve as a criterion to warrant the use of the simulator.

6. Identified and studied the problems simulating systems with feedback control.

7. Suggested manipulations to the problems so that developed tools can be applied such as rediagonalization and continuous-time approximation.

8. Applied the tools and methods to analyze SIM model v2.2, a task which was intractable before.

5.4 Recommendations for Future Work

The following are some of the interesting problems that require further investigations:

1. **Extension to nonlinear systems**. Although the linearity property is restricted only to LTI systems, the idea of parallelization is more general and it has been proved to be powerful for many other large-order problems.

2. **Sensitivity Analysis**. The simulator developed in this thesis allows more efficient time domain simulations than before. Hence the finite difference approximation to sensitivity matrix (Jacobian matrix) can be obtained with reasonable amount of computation and time with this new simulator. Sensitivity analysis is important to the design of systems since it points out the first direction one should try to make improvement.

3. **Further study at computer science level.** The analysis conducted in this thesis was at a user level. In order words, even if the underlying factors for improving computational efficiency were discovered, it did not necessarily contribute to the improvements of the simulator because of the limitation imposed by the available routines. A more fundamental study and manipulations at computer science level (e.g. memory management and diagonalization algorithm) can result in more significant gains in both efficiency and accuracy.

4. **First order hold discretization scheme.** First order hold scheme usually has better numerical result than zero order hold. The derivation of this scheme is included in Appendix B.

Appendix A

Modal Decomposition

In this chapter, the method of modal decomposition (a similarity transform that converts a dense A-matrix to a block diagonal one) will be discussed. The similarity transform of a dynamical system is as follows: Given a dynamical system

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{A.1}$$

where $x(t)$ is the state, $u(t)$ is the input, $y(t)$ is the output and the matrices A, B, C and D are real and of appropriate dimensions. Apply a change of coordinates of the state variables (i.e. $x = T\overline{x}$, where T is square and invertible), then the dynamical system in terms of $\overline{x}(t)$ will be

$$\begin{aligned} \dot{\bar{x}}(t) &= T^{-1}AT\bar{x}(t) + T^{-1}Bu(t) \\ y(t) &= CT\bar{x}(t) + Du(t). \end{aligned} \tag{A.2}$$

There are two ways to achieve the modal decomposition, provided that the A-matrix is diagonalizable.

Indexing. In this method, the transform matrix T of the similarity transform is a permutation matrix (i.e. a matrix obtained from full pivoting an identity matrix). As the name of the method suggests, the transform procedure in (A.2) can be avoided by merely reordering the state variables. For example, if the A-matrix of the system has the form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\omega_1^2 & 0 & -2\zeta_1\omega_1 & 0 \\ 0 & -\omega_2^2 & 0 & -2\zeta_2\omega_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ v_1 \\ v_2 \end{bmatrix}, \tag{A.3}$$

where x and v are given to remind the physical meanings of the state variables, namely position and velocity. ω and ζ are the natural frequency and damping ratio of the modes

respectively. The A-matrix in (A.3) is not diagonal or block diagonal. However, if the positions of v_1 and x_2 in the state vector are exchanged, then the resultant A-matrix is

$$\begin{bmatrix} \dot{x}_1 \\ \dot{v}_1 \\ \dot{x}_2 \\ \dot{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\omega_1^2 & -2\zeta_1\omega_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\omega_2^2 & -2\zeta_2\omega_2 \end{bmatrix} \begin{bmatrix} x_1 \\ v_1 \\ x_2 \\ v_2 \end{bmatrix}, \tag{A.4}$$

which is now block diagonal. Obviously, the situation in (A.3) can be generalized to the case in which the system is large-order. The B and C matrices of the transformed system can also be obtained by indexing with the same order. The advantage of this method is that it saves a significant amount of computation (recall from Chapter 2 that the cost of finding the eigenvalues and eigenvectors (the second method to be discussed) is $O(n_s^3)$). However, the applicability of the indexing technique is very limited. For example, if the A-matrix has more than 3 nonzero entries in any row or column, then the method fails. In general, the applicability of indexing should be motivated by the physical implication of the situation at hand. For example, the problem in (A.3) has all the information needed to construct the modal system, the only difficulty is that the state variables are in wrong order. Since indexing cannot in general be applied, the second method is more important in practice.

Eigenvalue Decomposition. In this method, the transform matrix T in (A.2) is the matrix whose columns are the eigenvectors of the original A-matrix. To see why $T^{-1}AT$ is diagonal, let's study the following informal proof. For the sake of simplicity, let's assume in the proof that the eigenvalues of the A-matrix are distinct (i.e. there is only one eigenvector associated with an eigenvalue). Let λ_i be an eigenvalue and $\underline{v}_i$ be the eigenvector associated with it. From the definition of eigenvalue, we have

$$A\underline{v}_i = \lambda_i \underline{v}_i.$$

Now if a matrix V is formed such that

$$V = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \cdots & \underline{v}_n \end{bmatrix},$$

where n is the number of eigenvalues. It can be verified that

$$\begin{aligned} AV &= \begin{bmatrix} \lambda_1\underline{v}_1 & \lambda_2\underline{v}_2 & \cdots & \lambda_n\underline{v}_n \end{bmatrix} \\ &= V\Lambda, \end{aligned}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. In general, the eigenvalues and the eigenvectors can be complex, even if the entries of the matrix is real. Hence the diagonalized system (A.2) will be complex, which is not very useful in applications. However, if the matrix is real, the complex eigenvalues (and the associated eigenvectors) must form complex conjugate pairs[1]. Taking advantage of the complex conjugate property of the eigenvalues and eigenvectors, a real transform matrix T can be obtained to achieve the similarity transform in (A.2) that results in a real transformed system. Let's study the following simplified proof. Let $A \in \mathbb{R}^{2\times 2}$ and it is diagonalized as follows:

$$A = \begin{bmatrix} v_1 & v_1^* \end{bmatrix} \begin{bmatrix} \alpha + j\beta & 0 \\ 0 & \alpha - j\beta \end{bmatrix} \begin{bmatrix} v_1 & v_1^* \end{bmatrix}^{-1}, \tag{A.5}$$

where α and β are the real part and imaginary part of the eigenvalue, $*$ denotes the operation of complex conjugate and $j = \sqrt{-1}$. Inserting an identity matrix

$$I = \begin{bmatrix} \frac{1}{2} & -j\frac{1}{2} \\ \frac{1}{2} & j\frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ j & -j \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ j & -j \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -j\frac{1}{2} \\ \frac{1}{2} & j\frac{1}{2} \end{bmatrix}$$

between the matrices in (A.5) and the following transform results

$$A = V'\Lambda' V'^{-1}, \tag{A.6}$$

where

$$V' = \begin{bmatrix} v_1 & v_1^* \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -j\frac{1}{2} \\ \frac{1}{2} & j\frac{1}{2} \end{bmatrix} = \begin{bmatrix} \text{Re}(v_1) & \text{Im}(v_1) \end{bmatrix}, \tag{A.7}$$

$$\Lambda' = \begin{bmatrix} 1 & 1 \\ j & -j \end{bmatrix} \begin{bmatrix} \alpha + j\beta & 0 \\ 0 & \alpha - j\beta \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -j\frac{1}{2} \\ \frac{1}{2} & j\frac{1}{2} \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}. \tag{A.8}$$

From (A.7), it is clear that the real transform matrix should be

$$T = \begin{bmatrix} \text{Re}(v_1) & \text{Im}(v_1) \end{bmatrix},$$

and the resultant system will remain real. The price for the real similarity transform is that the transformed system is no longer strictly diagonal, but block diagonal, as in (A.8). The similarity transform in (A.6) can be extended to systems with higher dimensionality since the procedure is the same and the resultant state space representation is called the real modal form. In the context of dynamical systems, the real part of an eigenvalue is $-\zeta_i\omega_i$

[1]Recall that solving the eigenvalues is equivalent to finding the roots of a polynomial.

and the imaginary part is $\pm\omega_i\sqrt{1-\zeta_i^2}$ for an underdamped mode, where ζ_i is the damping ratio and ω_i is the natural frequency of the i^{th} mode. Then the real modal form looks like

$$\Lambda' = \begin{bmatrix} -\zeta_i\omega_i & \omega_i\sqrt{1-\zeta_i^2} \\ -\omega_i\sqrt{1-\zeta_i^2} & -\zeta_i\omega_i \end{bmatrix}. \tag{A.9}$$

This form is preferred in numerical analysis because the entries of the A-matrix is more symmetrically placed, although the matrix is not symmetric. Another modal form results if yet another similarity transform is applied to (A.9). The transform matrix (for one mode) is

$$T_i = \begin{bmatrix} 1 & 0 \\ \frac{\zeta_i}{\sqrt{1-\zeta_i^2}} & \frac{1}{\omega_i\sqrt{1-\zeta_i^2}} \end{bmatrix}.$$

It can be verified that

$$\begin{aligned} T_i^{-1}\Lambda' T_i &= \begin{bmatrix} 1 & 0 \\ -\zeta_i\omega_i & \omega_i\sqrt{1-\zeta_i^2} \end{bmatrix} \begin{bmatrix} -\zeta_i\omega_i & \omega_i\sqrt{1-\zeta_i^2} \\ -\omega_i\sqrt{1-\zeta_i^2} & -\zeta_i\omega_i \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\zeta_i}{\sqrt{1-\zeta_i^2}} & \frac{1}{\omega_i\sqrt{1-\zeta_i^2}} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ -\omega_i^2 & -2\zeta_i\omega_i \end{bmatrix} \end{aligned} \tag{A.10}$$

The form in (A.10) is called 2^{nd} order modal form since the generic form of a 2^{nd} order system in s-domain is

$$\frac{k_i\omega_i}{s^2 + 2\zeta_i\omega_i s + \omega_i^2}, \tag{A.11}$$

where k_i is the modal gain. It is clear that (A.10) is the minimal realization of (A.11). The form in (A.10) is more revealing but it is in poorer numerical condition, particularly if low and high frequency modes coexist.

Two methods for modal decomposition were discussed in this section. For the indexing technique, the user is required to figure out the indexing sequence in the apparent cases. For eigenvalue decomposition, the implementation is coded in `ss2mod71.m`[2].

With the dynamical system transformed into block diagonal form, the simulator `new_lsim.m` can be applied to simulate the resultant system efficiently.

[2]This m-file is based on the script `ss2mod7.m` distributed within Space System Laboratory at MIT. An additional modal sorting particularly for the current purpose was included and thus the modified script is called `ss2mod71.m` for version 7.1.

Appendix B

First Order Hold

In Chapter 2 the derivation of the state transition formula was based on the assumption that the input to the plant is piecewise constant, which corresponds to the scheme called zero order hold (ZOH). Now, another discretization scheme is introduced. The scheme assumes that the input to the plant is piecewise linear over one sampling period and is called first order hold[1] (FOH). The derivation of this method is based on [30].

Suppose the plant is given in state space form

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right],$$

where the matrices are of appropriate dimensions. Let's begin with the Laplace transform of the first order holder. It has the form

$$\frac{e^{Ts} - 2 + e^{-Ts}}{Ts^2}, \tag{B.1}$$

where T is the sampling period and s is the complex frequency in the s-domain. It is clear that (B.1) is not causal in the s-domain. The block diagram of the equivalent discrete-time system is given in Figure B-1. In state space form, the block diagram in Figure B-1 can be

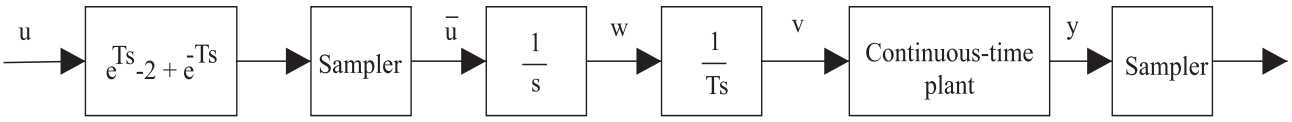


Figure B-1: Block diagram of a discretized plant with first order hold approximation.

represented as

$$\dot{x} = Ax + Bv$$

[1]It is a modified version of first order hold and is referred to as triangle hold in [30].

$$\begin{aligned} \dot{v} &= \frac{w}{T} \\ \dot{w} &= u(t+T)\delta(t+T) - 2u(t)\delta(t) + u(t-T)\delta(t-T). \end{aligned} \tag{B.2}$$

In matrix form, (B.2) can be given as

$$\begin{bmatrix} \dot{x} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ 0 & 0 & \frac{1}{T} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \bar{u}, \tag{B.3}$$

where the signals are given in Figure B-1. If we define the augmented state $\zeta = \begin{bmatrix} x & v & w \end{bmatrix}^T$ and the corresponding A-matrix F_T, then the one step state transition of ζ is

$$\zeta(kT+T) = e^{F_T T}\zeta(kT),$$

because $\overline{u}$ consists only of impulses at the sampling instants. If the matrix exponential of $e^{F_T T}$ is partitioned as

$$e^{(F_T T)} = \begin{bmatrix} \Phi & \Gamma_1 & \Gamma_2 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}, \tag{B.4}$$

then the samples of x are governed by

$$x(k+1) = \Phi x(k) + \Gamma_1 v(k) + \Gamma_2 w(k). \tag{B.5}$$

It can be verified from (B.2) that $v(k) = u(k)$ and that $w(k) = u(k+1) - u(k)$. Now if a new state is defined as

$$\xi(k) = x(k) - \Gamma_2 u(k),$$

then the state equation for ξ (from (B.5)) is

$$\begin{aligned} \xi(k+1) &= \Phi(\xi(k) + \Gamma_2 u(k)) + (\Gamma_1 - \Gamma_2)u(k) \\ &= \Phi\xi(k) + (\Gamma_1 + \Phi\Gamma_2 - \Gamma_2)u(k). \end{aligned} \tag{B.6}$$

The output equation for the plant is

$$\begin{aligned} y(k) &= Cx(k) + Du(k) \\ &= C(\xi(k) + \Gamma_2 u(k)) + Du(k) \\ &= C\xi(k) + (D + C\Gamma_2)u(k). \end{aligned} \tag{B.7}$$

In conclusion, the discretized system is of the form

$$\left[\begin{array}{c|c} A_d & B_d \\ \hline C_d & D_d \end{array}\right],$$

and the matrices are given as

$$\begin{aligned} A_d &= \Phi, \\ B_d &= \Gamma_1 + \Phi\Gamma_2 - \Gamma_2, \\ C_d &= C, \\ D_d &= D + C\Gamma_2, \end{aligned} \tag{B.8}$$

where Φ, Γ_1 and Γ_2 are defined by (B.4).

Although in the current version of `new_lsim.m` the first order hold discretization scheme is not employed, the extension to this scheme is straightforward because the idea of modal decoupling is valid no matter what kind of discretization scheme is chosen.

Bibliography

[1] P. Beran and W. Silva. Reduced-order modeling: New approaches for computational physics. In *39th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, January 2001.

[2] Christopher W. S. Bruner and Robert W. Walters. *Parallelization of the Euler Equations on Unstructred Grids.* Technical Report A97-32443, American Institute of Aeronautics and Astronautics, 1997.

[3] Alissa Clawson. *Digital to Analog Quantization Noise.* Technical report, Space System Laboratory, Massachusetts Institute of Technology, 2000.

[4] Chen C.T. *Linear System Theory and Design.* Oxford University Press, 1999.

[5] Carl de Boor. *A practical guide to splines*, volume I of *Applied mathematical sciences.* Springer Verlag, New York, 1 edition, 1978.

[6] Olivier L. de Weck. *Multivariable Isoperformance Methodology for Precision Opto-Mechanical Systems.* PhD thesis, Massachusetts Institute of Technology, September 2001.

[7] Haibo Dong and Xiaolin Zhong. A parallel high-order implicit algorithm for compressible Navier-Stokes equations. In *38th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, January 2000.

[8] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comp. Appl. Math.*, 6:19–26, 1980.

[9] Haier E. and Wanner G. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems.* Springer, 1996.

[10] Laila Mireille Elias. *A Structurally Coupled Disturbance Analysis Method Using Dynamic Mass Measurement Techniques, with Application to Spacecraft- Reaction Wheel Systems*. Master's thesis, Massachusetts Institute of Technology, March 2001.

[11] W. Gawronski. *Almost-Balanced Structural Dynamics*. Technical Report AIAA-97-1028, Jet Propulsion Laboratory, California Institute of Technology, 1997.

[12] Golub G.H. and Van Loan C.F. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, 3^{rd} edition, 1996.

[13] Strang Gilbert. *Introduction to Applied Mathematics*. Wellesley Cambridge Press, 1986.

[14] Andrew Grace. *Control system toolbox user's guide*. The MathWorks, Inc., 1996.

[15] C. Z. Gregory. *Reduction of Large Flexible Spacecraft Models Using Internal Balancing Theory*. *JGCD*, 7(6):17–32, December 1984.

[16] Kong Ha. *Sampled-data Control System Linear Analysis*. Technical Report 517, NASA Goddard Space Flight Center, 1998.

[17] F.B Hildebrand. *Advanced Calculus for Applications*. Prentice-Hall, 3rd edition, 1976.

[18] Jet Propulsion Laboratory, NASA, Pasadena, California, USA. *Integrated Modeling of Optical Systems User's Manual*, 4^{th} edition, September 1998.

[19] Van de Vegte John. *Feedback Control Systems*. Prentice Hall, 3rd edition, 1994.

[20] Little. John N. *Signal Processing Toolbox User's Guide*. The MathWorks, Inc., 1992.

[21] A. Laub. Computation of balancing transformations. *Proceedings of JACC*, 1(FA8-E), 1980.

[22] A.J. Laub, M.T. Heath, C.C Paige, and R.C Ward. Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms. *IEEE Trans. Automatic Control*, (AC-32):115–122, 1987.

[23] Peiman G. Maghami and Sean P. Kenny. Algorithms for efficient computation of transfer functions for large order flexible systems. Technical Report 206949, NASA Langley Research Center, March 1998.

[24] B. C. Moore. Principal component analysis of linear systems: Controllability, observability, and model reduction. *ITAC*, AC-26:17–32, February 1981.

[25] A.V. Oppenheim, R.W. Schafer, and Buck. *Discrete-time Signal Processing*. Prentice Hall, 2nd edition, 1999.

[26] Sebastian Reich. Multiple time scales in classical and quantumclassical molecular dynamics. *Journal of Computational Physics*, 1999.

[27] William McC. Siebert. *Circuits, Signals, and Systems*. The MIT Electrical Engineering and Computer Science Series. The MIT Press, 1986.

[28] Robert E. Skelton, Peter C. Hughes, and Hari B. Hablani. Order reduction for models of space structures using modal cost analysis. *AIAA Journal of Guidance*, 5(4):351–357, July-Aug 1982. AIAA 82-4194.

[29] David W. Miller, Olivier L. de Weck and Scott A. Uebelhart. *Integrated Dynamics and Controls Modeling for the Space Interferometry Mission (SIM).* In *IEEE Aerospace Conference*, Big Sky, Montana, USA, March 2001.

[30] Franklin G.F., Powell J.D. and Workman M.L. *Digital Control of Dynamic Systems*. Addison Wesley, 2^{nd} edition, 1990.

[31] Myles L. Baker, D. L. Mingori and Patrick J. Goggin. Approximate subspace iteration for constructing internally balanced reduced order models of unsteady aerodynamic systems. In *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference and Exhibit,37th.*, Salt Lake City, UT, USA, April 1996.

[32] Kailath Thomas. *Linear Systems*. Prentice-Hall information and system sciences series. Englewood Cliffs, N.J. : Prentice-Hall, 1980.

[33] Chen Tongwen and Bruce Francis. *Optimal Sampled-Data Control Systems*. Communications and Control Engineering Series. Springer, London, 1995.

[34] Lloyd N Trefethen. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[35] Scott A. Uebelhart. *Conditioning, Reduction, and Disturbance Analysis of Large Order Integrated Models of Space-Based Telescopes*. Master's thesis, Massachusetts Institute of Technology, February 2001.

[36] J.K. White. *Introduction to Numerical Simulation*. 16.910J. MIT classnotes, 2001.

[37] K. Willcox and J. Peraire. Balanced model reduction via the proper orthogonal decomposition. In *15th AIAA Computational Fluid Dynamics Conference*, Anaheim, CA, USA, June 2001.

[38] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice-Hall, Inc., 1996.