FAST TIME DOMAIN SIMULATION FOR LARGE-ORDER LINEAR TIME INVARIANT SYSTEMS

Kin Cheong Sou
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, U.S.A.
email: kcsou@mit.edu

Olivier L. de Weck
Department of Aeronautics and Astronautics
Engineering Systems Division (ESD)
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, U.S.A.
email: deweck@mit.edu

ABSTRACT
Time domain simulation can be time consuming and cause memory saturation problems when systems get large, sampling rates are high and input time histories are long. In this paper, an efficient simulation scheme is presented to simulate large order continuous-time linear time-invariant (LTI) systems. The A matrix (assumed to be block-diagonalizable) of the system is first diagonalized. Then, subsystems of manageable dimensions and bandwidth are formed and multiple sampling rates can be chosen to associate with the subsystems. Each subsystem is then discretized using a O(n_s) discretization scheme, where n_s is the number of state variables. Next, a sparse matrix recognizable O(n_s) discrete-time system solver (i.e. matrix-vector product solver) is employed to compute the history of the state and the output. Finally, the response of the original system is obtained by superposition and interpolation of the subsystem responses. The simulation scheme is shown to be superior in the case of medium and large order systems.

KEY WORDS
Time Domain, Simulation, State Transition, Linear Time Invariant Systems

Nomenclature

A, B, C, D	=	State space matrices
BZ	=	Block size of a subsystem
DSF	=	Downsampling factor
EM	=	Ending mode
SM	=	Starting mode
SS	=	Subsystem
TM	=	Total number of modes
T	=	CPU time
T	=	Sampling time period
m	=	Number of input channels of a system
n	=	Number of samples / Matrix dimension
n_s	=	Number of state variables
p	=	Number of output channels of a system
u	=	Control input
v	=	Discrete-time signal
w	=	Disturbance input
x	=	State vector
y	=	Measurement output
z	=	Performance output
σ	=	Root-mean-square

1 Introduction

Simulation is an essential tool for the design of complex engineering systems, as the models of these systems become more and more complex. Model dimension (or size/order) is a delicate issue in systems design because of the tradeoff between model fidelity and the number of designs explored within a limited time budget, $T_{tot} = N \times T_{cpu}$, where N is the number of simulations or designs explored and T_{cpu} is the runtime of a single simulation. The dilemma is shown in Fig. 1. It is clear from Fig. 1 that the more efficient a

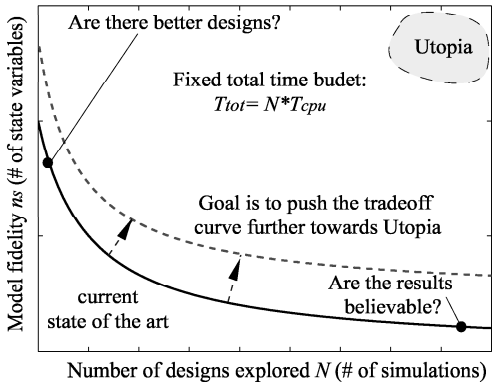

Figure 1. Notional tradeoff curve between number of designs explored and model fidelity. The solid line is the curve allowed by current techniques, based on a $T_{cpu} \propto n_s^3$ scaling relationship, see de Weck et al.[1]. The dashed line is the objective of this work. The upper right corner is the utopia point.

simulation is for a given level of model fidelity, the better a position system designers find themselves in. In Gutierrez [2] and de Weck et al. [1], three methods for performance evaluation of dynamic systems are summarized: 1) Time domain simulation, 2) Frequency domain analysis and 3) Lyapunov analysis. In these papers, improvements for frequency domain and Lyapunov methods were proposed with the derivation of new algorithms such as `newlyap.m`, but the problem of time domain simulation remained unsolved. Particularly challenging time domain simulations are caused by large dimensionality of the state space system ($n_s > 500$) and/or large model dynamic bandwidth as it occurs in engineering systems or molecular dynamics [3]. The problem can be relieved without parallelization, if not entirely solved, by the capability of efficient time domain simulation of large order systems, which is the topic of this paper. Computationally efficient algorithms can be viewed as a complement to model reduction techniques such as the ones recently proposed by de Weck et. al. [1], Willcox and Peraire [4] as well as Beran and Silva [5].

2 Time Domain Simulation Problem

The time domain simulation problem of an LTI system can be defined as follows: Given a system in (1)

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

where $x(t)$ is the state, $u(t)$ is the input, $y(t)$ is the output and the matrices are of appropriate dimensions. The solution to this problem is the unique $y(t)$, given an external input $u(t)$ and initial conditions $x(t_0)$. There are at least two ways to solve the problem: 1) Standard ordinary differential equation (ODE) solvers like Runge Kutta [6], and 2) The state transition formula in (2)

$$x(t) = \underbrace{e^{A(t-t_0)}x(t_0)}_{x_H} + \underbrace{\int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)d\tau}_{x_p}, \tag{2}$$

where t_0 denotes the initial time instant, τ is a dummy variable, $e^{A(t-\tau)}$ is the matrix exponential of the matrix $A(t-\tau)$ and $u(t)$ is the external input to the system, x_H and x_P denote the homogenous and particular solutions, respectively. The state transition method just mentioned is equivalent to transforming the original problem (1) into its discretized version in (3), provided that a further assumption is made on $u(t)$ (e.g. zero order hold (ZOH)).

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

where, (3)

$A_d = e^{AT}$,

$B_d = \int_{KT}^{(K+1)T} e^{A(KT+T-\tau)}Bd\tau = \int_0^T e^{A\tau}Bd\tau$,

and T is the sampling period and $n \in \mathbb{Z}$.

The state transition method serves the current problem better in that it requires less computation and it maps left half s-plane poles to inside the unit circle in the z-plane, no matter how large the discretization time step, T, is. Nevertheless, the benefits of the state transition method do not come for free in that the following problems must be addressed: The first problem is the computational expense of e^{AT} in (3). The cost of this operation is O(n_s^3) with n_s being the number of state variables [7]. The second problem is memory saturation, if the computation requires the whole history of states before the history of the output can be computed (see `lsim.m` for such an algorithm [8]), in that case the simulation might halt because of memory saturation. The solutions to these problems will be given in Section 3.

3 Fast Time Domain Simulation Algorithm

In this section, the steps and implementation issues of the presented algorithm will be addressed. Before the details are discussed, an overview of the algorithm is given by the flowchart in Fig. 2. In this flow chart, the circles correspond to data such as the original system A, B, C, D matrices, the external input and the computed response (output). Each block represents an operation or process with the actual MATLAB implementation label next to it (The name of the m-file is in italic font).

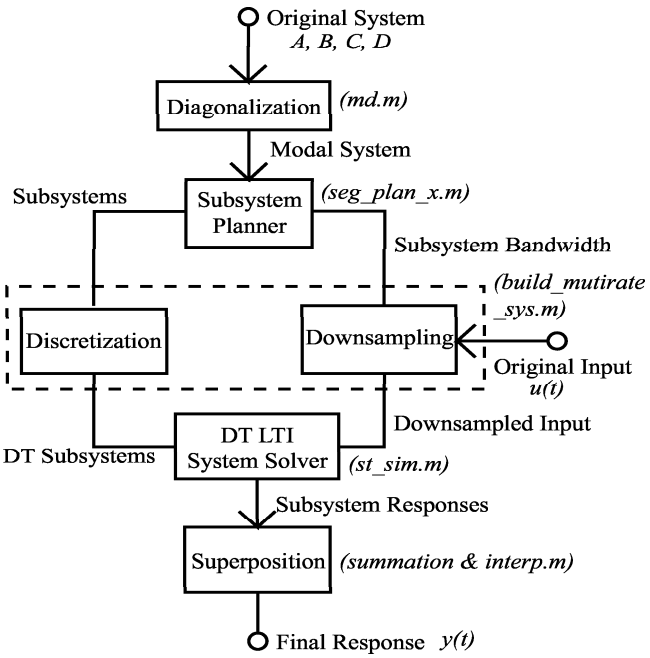

Figure 2. Flow chart of the fast time domain simulation algorithm, `newlsim.m`.

3.1 Diagonalization

Diagonalization is a similarity transform of the original system into a system that has a block diagonal A matrix. That is,

$$A = S\Lambda S^{-1},$$

where S is an invertible similarity transform matrix and Λ is a real block diagonal matrix. This similarity transform can be an eigenvalue decomposition or state variable reordering, whereby the latter is less expensive. Additionally, the corresponding subroutine of the presented simulation scheme sorts the modes of the diagonalized system in ascending natural frequencies. This diagonalization step is necessary in order to enable the following implementations:

- *Decoupling the dynamics and forming fictitious subsystems.*
- *Applying multiple sampling rates.*
- *Exploiting the sparsity resulting from the diagonal structure of the A matrix.*

3.2 Subsystem Planning

The subsystem planning subroutine is the key decision element in the algorithm. There are two considerations for the subsystem planning. The first issue is the size (number of state variables) of each subsystem. According to Sou[7] the approximate cost is[1],

$$\text{FLOPS} = 2 \times (2 + m + p) \times n_s \times n \tag{4}$$

where m is the number of input channels, p is the number of output channels, n_s is the number of state variables and n is the number of samples to be processed. This is in contrast to non-sparse continuous time system simulation, where the cost is proportional to the number of states to the third power[1]. Nevertheless, simulating subsystems that are too large and too small is not efficient because of memory saturation and size independent overhead, respectively.

The other issue is the sampling rate for each simulation. The minimum sampling rate is given by Nyquist's sampling theorem but it is usually insufficient for computer simulations. As a rule of thumb, the sampling rate should be four to ten times the system bandwidth. The issue of appropriate sampling rate selection has been extensively discussed by Åström [9] among others. A flow chart of this subroutine is given in Fig. 3.

In this flowchart, the f(DSF) is implemented as a bisection process that determines the ending mode (EM) of each subsystem.

[1]The FLOPS due to feedthrough are ignored.

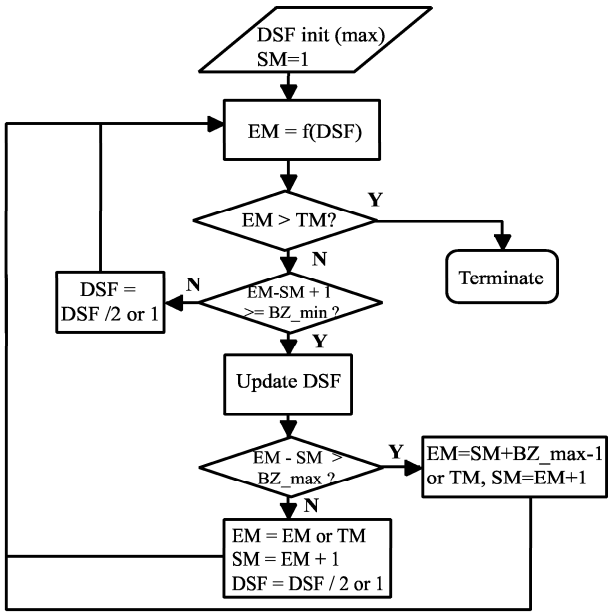

Figure 3. Flow chart of the subsystem planning subroutine. DSF denotes downsampling factor (the ratio between the original sampling rate and the reduced sampling rate). SM is the starting mode of the subsystem. EM is the ending mode of the subsystem. TM is the total number of modes of the original system. This serves as a termination criterion. BZ is the currently chosen block size of each subsystem.

3.3 Discretization

As can be seen in (3), the bottleneck of discretization is the matrix exponential. Fortunately the diagonalization described earlier relieves the problem. By exploiting the sparsity of the block diagonal A matrix structure, a O(n_s) matrix exponential algorithm can be realized. This is obviously seen if the matrix exponential A_d in (3) is expressed as follows (see Chen [10], for example):

$$A_d = e^{AT} = I + AT + \frac{A^2T^2}{2} + \frac{A^3T^3}{3!} + \cdots. \tag{5}$$

By noticing the fact that

$$A^nT^n = \begin{bmatrix} A_1T & 0 & \cdots \\ 0 & A_2T & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}^n = \begin{bmatrix} A_1^n & 0 & \cdots \\ 0 & A_2^n & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} T^n, \tag{6}$$

where $A_i \in \mathbb{R}^{2\times 2}\ \forall\, i \in \{1, 2, \ldots\}$ and applying (6) to (5), the following equality holds:

$$\exp\left(\begin{bmatrix} A_1T & 0 & \cdots \\ 0 & A_2T & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}\right) = \begin{bmatrix} e^{A_1T} & 0 & \cdots \\ 0 & e^{A_2T} & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}. \tag{7}$$

The advantage of fast discretization comes with the expense of the diagonalization step itself, which is also O(n^3). Fortunately, the computation of eigenvalues is usually more efficient than that of the matrix exponential. Other references for discrete time systems are by Yamamoto [11], Chen and Francis [12] as well as Oppenheim et. al [13].

3.4 Downsampling

Downsampling the input is required whenever the assigned sampling rate of a particular simulation is lower than that of the input (this is true if the multiple sampling rates strategy is employed). The straightforward way to achieve the goal is to low-pass filter the signal and then downsample it. That is,

$$x_d[n] = x[Mn],$$

where $x[n]$ has been low-pass filtered and M is the downsampling factor.

In addition to the direct method just discussed, there is another way to downsampling. Recall the state transition formula (or the state equations for the discretized system),

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n]. \end{aligned} \tag{8}$$

The first equation in (8) is certainly satisfied at the $n+1$ instant,

$$x[n+2] = A_d x[n+1] + B_d u[n+1]. \tag{9}$$

If (8) is substituted into in (9), then the following results

$$\begin{aligned} x[n+2] &= A_d \left\{ A_d x[n] + B_d u[n] \right\} + B_d u[n+1] \\ &= A_d^2 x[n] + A_d B_d u[n] + B_d u[n+1]. \end{aligned} \tag{10}$$

Equation (10) can be generalized for N steps,

$$x[n+N] = A_d^N x[n] + A_d^{N-1} B_d u[n] + \cdots + B_d u[n+N-1] \tag{11}$$

If the following new matrices are defined

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

then the N-step propagation version of (8) is

$$\begin{aligned} x[n+N] &= \underline{A_d} x[n] + \underline{B_d}\,\underline{u}[n] \\ y[n] &= \underline{C} x[n] + \underline{D}\,\underline{u}[n]. \end{aligned} \tag{12}$$

By employing a long time step state transition, the calculation of the unwanted intermediate states and outputs can be avoided. This downsampling scheme essentially downsamples the output instead of the input and the accuracy is much better than the first direct approach. However, the efficiency gain achieved by the second method is less than that by the first method. Compare the FLOPS for the direct downsampling (13) and those of the second method (14):

$$\text{FLOPS} = 2 \times \frac{(2+m+p) \times n_s \times n}{\text{DSF}}, \tag{13}$$

$$\text{FLOPS} = 2 \times \frac{(2+DSF \times m+p) \times n_s \times n}{DSF}. \tag{14}$$

Here $DSF \geq 1$ is the downsampling factor. The meanings of other parameters in (13) and (14) are referred back to (4). In conclusion, the second method is more conservative. The current version of `newlsim.m` adopts the second method, also referred to as lifting, if downsampling is to be realized.

3.5 Simulation, Interpolation and Superposition

With the subsystems formed and discretized, the burden on the simulator (the actual code that computes the states and outputs) is lighter compared to the original ODE problem and it now becomes a much simpler problem of matrix-vector multiplication (cf. (3)). Taking into account the sparsity of the A matrix, the matrix-vector multiplication (state transition) can be realized in O($n_s n$) (n_s is the number of state variables and n is the number of samples to be simulated) FLOPS.

Interpolation is needed whenever the output is downsampled. Because the downsampling instants are evenly spaced, the downsampling can be viewed as the resampling of discrete-time signals and efficient interpolation methods in signal processing (i.e. inserting zeros and then low-pass filtering) can be employed. In fact, `newlsim.m` applies the MATLAB command `interp.m` [14], which does exactly this.

With the responses of the subsystems computed, it is finally possible to form the response of the original system due to the original input. This is allowable because of the linearity property of LTI systems.

The commands mentioned above are very MATLAB specific because the current version of `newlsim.m` is implemented in MATLAB. Nevertheless, the idea to take advantage of problem specific insights and structure extends naturally to more general platforms.

4 Simulation Results

In this section, some application examples are given to show the potential value of the `newlsim.m` algorithm. The first example computes the performance RMS values of randomly generated stable SISO systems of different di-

mensions, n_s, driven by randomly generated input signals[2]. Similar to previous work by de Weck et al. [1], two methods are compared: Frequency domain method and Time domain method. For the time domain method, the input signal is 10^5 samples long and for the frequency domain method, 10^5 frequency points (single sided) are computed. The results are summarized in Table 1.

Table 1. Comparison between frequency domain and time domain methods for randomly generated systems of different size. N/A indicates memory saturation.

n_s	results	`freq`	`lsim`	`newlsim`
50	T_{cpu}[s]	0.2960	1.3750	0.6880
	σ	1.714E-6	1.722E-6	1.722E-6
100	T_{cpu}[s]	0.6410	4.5310	1.0150
	σ	1.167E-5	1.187E-5	1.187E-5
200	T_{cpu}[s]	2.0000	71.078	2.3900
	σ	1.516E-5	1.463E-5	1.463E-5
500	T_{cpu}[s]	15.3280	N/A	15.2030
	σ	3.006E-5	N/A	3.061E-5
1000	T_{cpu}[s]	104.25	N/A	96.0320
	σ	5.452E-5	N/A	5.228E-5
1500	T_{cpu}[s]	425.3	N/A	382.7
	σ	1.346E-5	N/A	1.334E-5
2000	T_{cpu}[s]	1046.4	N/A	934.4
	σ	2.869E-5	N/A	2.733E-5

In this table, n_s is the number of state variables. T_{cpu} is the CPU time (in seconds) of each computation and σ is the RMS value of each performance output (the actual unit is not of interest here). Methods: `freq` denotes the frequency domain method, `lsim` is the standard time domain simulation method provided by MATLAB. Finally, `newlsim` is the proposed time domain simulation method. Note that `freq` is the fast implementation of the frequency domain method, see de Weck et al. [1]. Note also that the time for `newlsim` includes the time to diagonalize. Since new data is generated in each case with a different n_s, the RMS values of different n_s cases differ accordingly and should be not compared.

The main point to illustrate here is that the performance RMS values computed by the three different methods are quite close to each other. As shown in Table 1, time domain methods are generally not as efficient as other methods for small systems ($n_s < 100$). However, the situation changes when systems get larger ($n_s > 500$). The reason for this trend is that size-independent overhead of the time domain method is more significant in small system cases. It should also be noted that the computation time of the frequency domain and time domain methods varies with the number of samples evaluated. The reason why the results from `lsim` are unavailable (N/A) is that the computer ran out of memory, which means that `lsim` is not very suited for large-order systems simulation. The above example shows that `newlsim` can achieve efficiency similar to the fast implementations of other methods (e.g. frequency domain) with acceptable accuracy when computing RMS values.

Computing output RMS values does not fully exemplify the advantage of `newlsim`. A time domain simulation scheme should be compared with another time domain simulation scheme. Therefore the MATLAB simulator, `lsim.m`, is chosen as a reference in the following example. In the example, a number of randomly generated systems with different sizes are simulated in the time domain with `lsim.m` and `newlsim.m`. The computation time of each simulation is given in Figure 4. The time responses of

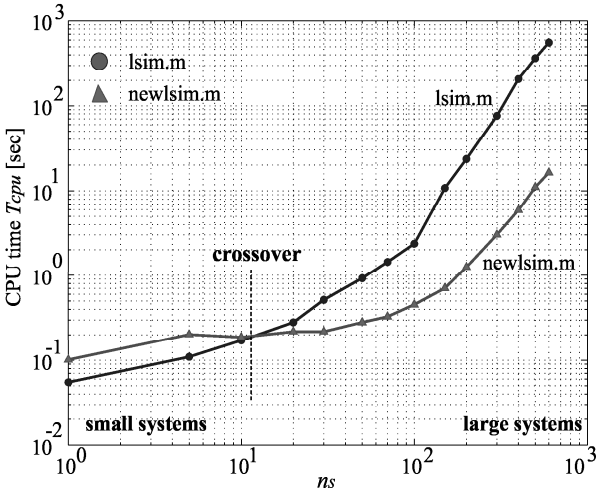

Figure 4. Logarithmic plot of CPU times by `lsim.m` and `newlsim.m` versus number of state variables. Circle: `lsim.m` time. Triangle: `newlsim.m` time. Maximum savings factor $T_{lsim}/T_{newlsim} \simeq 50$. Crossover at around $n_s = 12$ state variables.

a sample system by `lsim.m` and `newlsim.m` are shown in Figure 5. The example shows that `newlsim.m` is more efficient than `lsim.m`, while the error is insignificant.

5 Summary

Time domain simulation is an important technique for multidisciplinary design, analysis and optimization of dynamic systems. Unfortunately, as model fidelity and size get large one experiences excessive computation times and memory saturation problems. In this paper, the simulation scheme, `newlsim.m`, based on a block diagonaliza-

[2]The test machine is a Pentium 4 PC with a 1.5GHz CPU and 128MB RAM. Any other computations in this report were performed on the same machine

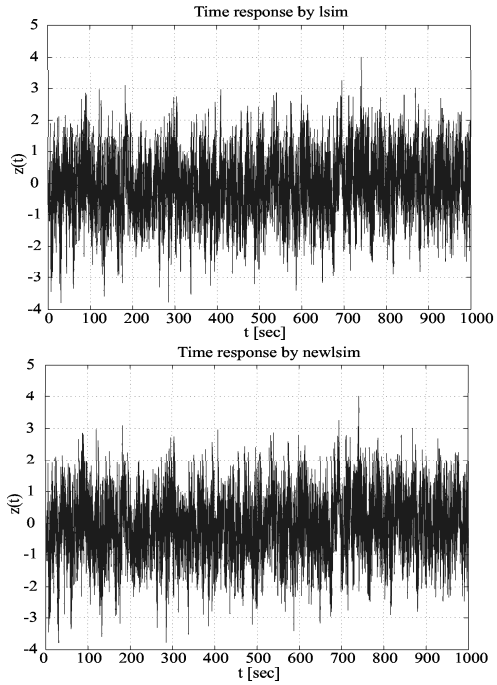

Figure 5. Time responses for a $n_s = 200$ state system by `lsim.m` - $T_{cpu} = 71.1$ [sec] - and `newlsim.m` $T_{cpu} = 2.4$ [sec]. Both the system and the external input are randomly generated.

tion pre-processing, is presented in response to this challenge. The targeted systems are large-order, diagonalizable continuous-time LTI systems. It has been found that diagonalization provides three benefits (dynamics decoupling, fast discretization and multiple sampling rates) that facilitate the simulation. In conjunction with the block diagonalization structure of the resultant A matrix, it has been shown that a sparse matrix recognizable state transition must be employed in order to achieve the O($n_s n$) state transition by taking advantage of the resultant sparsity.

Recommendations for future work include extensions of the algorithm to time-varying and weakly non-linear systems as well implementation of distributed computation of the subsystem responses on parallel computers.

Acknowledgement

This research was funded by the NASA Jet Propulsion Laboratory under grant No. JPL 91123 and was monitored by Dr. Ipek Basdogan. The authors thank Prof. Wallace Vander Velde of MIT for his helpful comments.

References

[1] de Weck, O., Uebelhart, S., Gutierrez, H., and Miller, D., "Performance and Sensitivity Analysis for Large Order Linear Time-Invariant Systems," *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, Georgia, USA, September 2002.

[2] Gutierrez, H. L., *Performance Assessment and Enhancement of Precision Controlled Structures During Conceptual Design*, Ph.D. thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 1999.

[3] Reich, S., "Multiple Time Scales in Classical and QuantumClassical Molecular Dynamics," *Journal of Computational Physics*, Vol. 151, No. 1, May 1999, pp. 49–73.

[4] Willcox, K. and Peraire, J., "Balanced Model Reduction via the Proper Orthogonal Decomposition," *15th AIAA Computational Fluid Dynamics Conference*, Anaheim, CA, USA, June 2001.

[5] Beran, P. and Silva, W., "Reduced-Order Modeling: New Approaches for Computational Physics," *39th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, January 2001.

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

[7] Sou, K. C., *Fast Time Domain Simulation for Large Order Hybrid Systems*, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, May 2002, SSL Report # 11-02.

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

[9] Åström, K. J. and Wittenmark, B., *Computer-Controlled Systems: Theory and Design*, Prentice-Hall, Inc., 1997.

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

[11] Yamamoto, Y., "A function space approach to sampled data control systems and tracking problems," *IEEE Transactions on Automatic Control*, Vol. 39, No. 4, April 1994, pp. 703–713.

[12] Chen, T. and Francis, B., *Optimal Sampled-Data Control Systems*, Communications and Control Engineering Series, Springer, London, 1995.

[13] Oppenheim, A., Schafer, R., and Buck, *Discrete-time Signal Processing*, Prentice Hall, 2nd ed., 1999.

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