INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2005; **63**:681–708
Published online 9 March 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1294

Fast time-domain simulation for large-order linear time-invariant state space systems

Kin Cheong Sou and Olivier L. de Weck*, †, ‡, §

Department of Aeronautics and Astronautics, Engineering Systems Division, MIT, Cambridge, MA 02139, U.S.A.

SUMMARY

Time-domain simulation is essential for both analysis and design of complex systems. Unfortunately, high model fidelity leads to large system size and bandwidths, often causing excessive computation and memory saturation. In response we develop an efficient scheme for large-order linear time-invariant systems. First, the A matrix is block diagonalized. Then, subsystems of manageable dimensions and bandwidth are formed, allowing multiple sampling rates. Each subsystem is then discretized using a $O(n_s)$ scheme, where n_s is the number of states. Subsequently, a sparse matrix $O(n_s)$ discrete-time system solver is employed to compute the history of the state and output. Finally, the response of the original system is obtained by superposition. In practical engineering applications, closing feedback loops and cascading filters can hinder the efficient use of the simulation scheme. Solutions to these problems are addressed in the paper. The simulation scheme, implemented as a MATLAB function `fastlsim`, is benchmarked against the standard LTI system simulator `lsim` and is shown to be superior for medium to large systems. The algorithm scales close to $O(n_s^2)$ for a set of benchmarked systems. Simulation of a high-fidelity model ($n_s \approx 2200$) of the Space Interferometry Mission spacecraft illustrates real world application of the method. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: time-domain simulation; LTI systems; multiple-sampling rates; discretization; down-sampling; lifting

1. INTRODUCTION

1.1. Background

Simulation is an essential tool for the design and analysis of complex engineering systems. Computational challenges arise as the models of these systems become more and more complex. For linear time-invariant systems complexity manifests itself both as model size (order

*Correspondence to: O. L. de Weck, MIT, 77 Massachusetts Ave, 33-410, Cambridge, MA 02139, U.S.A.
†E-mail: deweck@mit.edu
‡Assistant Professor.
§Senior member AIAA.

Contract/grant sponsor: NASA Jet Propulsion Laboratory; contract/grant number: 91123

Received 3 September 2003
Revised 19 May 2004
Accepted 1 December 2004

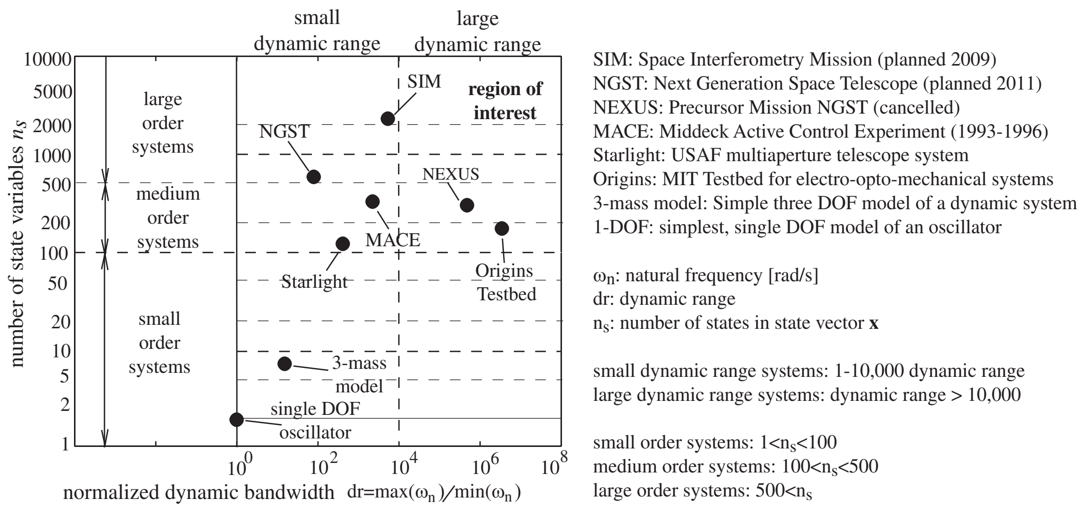


Figure 1. Classification of LTI systems according to size and bandwidth.

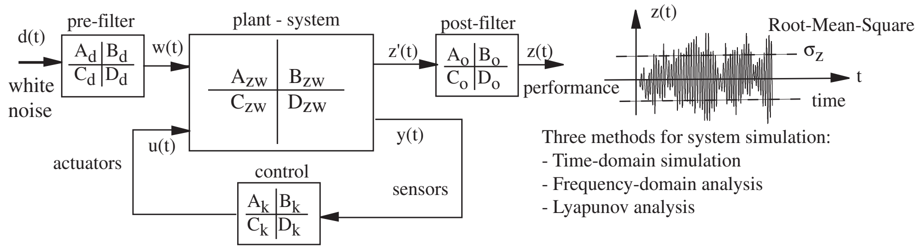


Figure 2. Typical dynamic system simulation, resulting in an estimate of $z(t)$.

of the system or number of states, n_s) as well as model dynamic range (ratio $\omega_{n,\max}/\omega_{n,\min}$). Figure 1 shows some sample systems positioned in terms of model dynamic range and model size. Dynamic range ('bandwidth') is defined as the ratio of the highest to lowest natural frequency of the system.

Generally, we are interested in modelling the behaviour of these systems in terms of their performance outputs, $z(t)$, subject to disturbance inputs $w(t)$, control inputs, $u(t)$ and sensor measurements, $y(t)$, see Figure 2. We assume that the state space matrices A, B, C, D are known.

A popular measure of performance of a dynamic system is the root-mean-square (RMS) statistic, $\sigma_z = \left[1/T \int_0^T z^2(t)\,\mathrm{d}t\right]^{1/2}$. This is valid under the assumption of stationary disturbance sources, $w(t)$, and time-invariant system dynamics (A, B, C, D). We have previously identified three methods for estimating σ_z: time-domain simulation, frequency-domain analysis and Lyapunov analysis. These three approaches are briefly summarized in Appendix A.

1.2. Motivation and previous work

Previous work has been conducted on comparing both the accuracy and computational expense of the three approaches [1–3]. This work also suggested improvements for speed-up of frequency-domain and Lyapunov analysis, respectively. These were based in part on using balanced model reduction (see Reference [4] for balanced reduction with *a priori* error bounds). However, the problem of computational expense for time-domain simulation remained unsolved. While frequency-domain analyses and Lyapunov analysis can provide critical performance metrics such as steady-state RMS values of performances, they cannot provide information on the transient response of the system, which is sometimes required (e.g. in designing control systems). Also, while model reduction can result in reduced systems with high accuracy, the method suffers from the fact that *a priori* decisions must be made as to the level of the reduction. Model reduction is being pursued in a variety of fields, see Willcox *et al.* [5] and Beran [6], but is fundamentally different from the divide-and-conquer approach advocated here. The focus of this paper is on mitigating problems with time-domain simulation for large LTI systems.

We shall first conduct a small numerical experiment to estimate the increase in computational cost for all three methods, as a function of system size, n_s, and dynamic range dr. We first generate random SISO state space systems (A, B, C, D) with natural frequencies uniformly, but randomly, distributed between 1 Hz and dr Hz. The damping ratio of the system modes shall be uniformly distributed between $10^{-2} < \zeta_i < 1$. Further, let the system be represented in second-order modal form: $A = [0 \;\; I; -\Omega^2 - 2\Omega Z]$ with $\Omega = \mathrm{diag}(\omega_i)$ and $Z = \mathrm{diag}(\zeta_i)$. The structure of the A-matrix and time- and frequency-domain simulation results for the baseline case with $n_s = 100$ and $dr = 100$ are shown in Figure 3 and Table I.

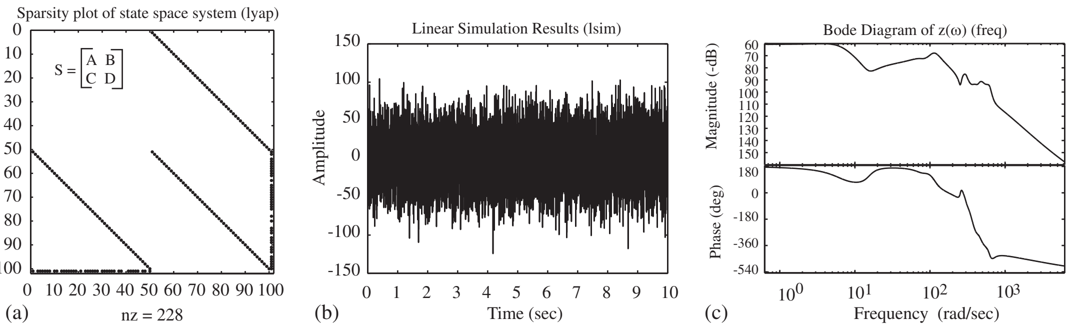


Figure 3. (a) State space system sparsity matrix, non-zero entries are shown; (b) $z(t)$ from time simulation using MATLAB's `lsim` function; and (c) $z(\omega)$ in frequency domain.

Table I. Baseline experiment with $n_s = dr = 100$.

Method	T_{CPU} (ms)	σ_z, 10^{-3}	Error (%)
`time`	661	2.312	1.59
`freq`	371	2.289	0.61
`lyap`	300	2.326	0

Table II. Numerical experiment with random state space systems, T_{CPU} [s].

dr	10	20	50	100	200	500	1000	2000
`time`	0.13	0.11	0.26	0.33	0.66	1.70	3.39	77.9
`freq`	0.10	0.09	0.18	0.34	0.69	1.74	5.02	10.69
`lyap`	0.33	0.28	0.28	0.28	0.28	0.28	0.28	0.99
n_s	10	20	50	100	200	500	1000	
`time`	0.36	0.08	0.19	0.40	1.27	35.8	191.5	
`freq`	0.02	0.08	0.018	0.35	0.91	6.32	45.49	
`lyap`	0.03	0.03	0.050	0.29	2.00	41.85	549.6	

Table I shows that the prediction error for σ_z relative to the Lyapunov method is small ($<2\%$) and that the computation times are also small (<1 s) and similar for all three approaches.

Next, we repeat the experiment and scale both the dynamic range (dr) as well as the system size (n_s) with results shown in Table II. The computer used throughout this research is a 32-bit Pentium 4 with a 1.8 GHz processor and 256 MB of RAM.

We clearly see that T_{CPU} scales linearly with dynamic range, dr above $n_s = 20$ for both the time- and frequency-domain methods. This is due to the fact that the time horizon T and time step, Δt are chosen as

$$T = 10\left[\frac{2\pi}{\min(\omega_n)}\right] \quad \text{and} \quad \Delta t = \frac{1}{10}\left[\frac{2\pi}{\max(\omega_n)}\right] \tag{1}$$

Note, also that the cost of the Lyapunov method is independent of dr. The linear scaling ceases to hold true once RAM memory is saturated ($dr = 2000$, $T_{\mathrm{CPU}} = 77.9$) for the time-domain method. Once swapping and access to virtual memory are required simulation becomes very inefficient.

The issue of scaling is more serious in the case of increasing system size n_s. Table II shows that the cost of all three methods increases proportional to $\propto n_s^3$. This is shown more clearly in Figure 4(a). Model size, n_s, is a delicate issue in system simulation because of the tradeoff between model fidelity and the number of design solutions or disturbance scenarios explored within a limited computational time budget. The dilemma is shown in Figure 4(b). On the one hand, one may choose high model fidelity (n_s large), but only be able to explore a limited number of scenarios. On the other hand, one may choose a small model (n_s small) and carry out many simulations, but be left wondering about the validity of the results.

It is clear from Figure 4(b) that the more efficient a simulation is, for a given model size and bandwidth, the more scenarios can be evaluated. The two curves in the figure should represent the same amount of total time budget: $T_{\mathrm{total}} = N \times T_{\mathrm{CPU}} = N \times \left(\alpha \times n_s^{\beta}\right)$. The curves scale roughly as $1/n_s^3$. An improvement in α or β (dashed line) could increase the product of $N \times n_s$, thus providing better simulation capability. This assumes that time-domain simulation accuracy is not sacrificed. The upper right corner is the ultimate desire. In addition, efficient simulation can facilitate the performance evaluation step of multidisciplinary design optimization (MDO), whose prospects and significance have been reported by Giesing *et al.* [7], Sobieski *et al.* [8] and Anderson [9] among others.

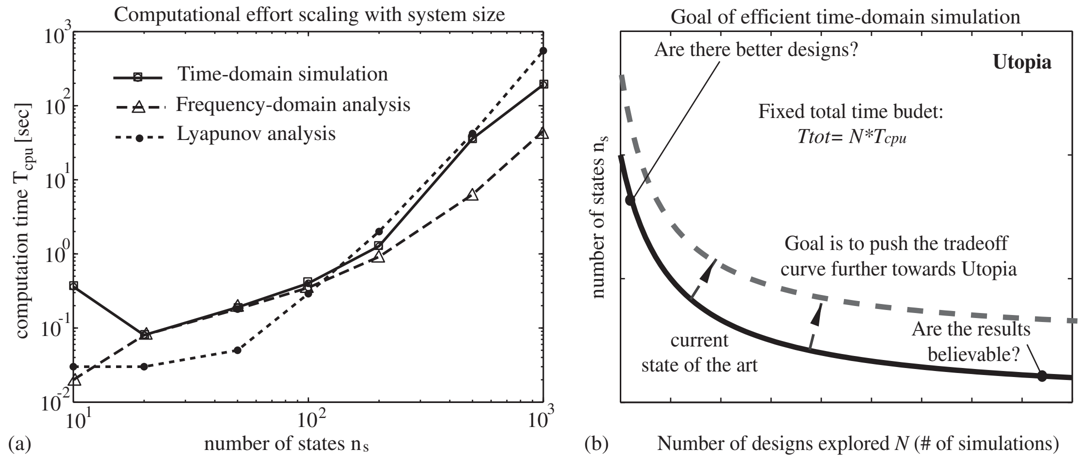


Figure 4. (a) T_{CPU} scaling with model size n_s and method; and (b) tradeoff curve between number of simulations (N) and model fidelity (n_s).

The proposed simulation algorithm, `fastlsim`, first decouples the original dynamical system by implementing a block diagonalization[¶] of the A matrix. Then, it forms fictitious subsystems with lower, and thus manageable dimensions and narrower bandwidths. After that, the subsystems are discretized so that efficient computation of state transition can be realized. Finally, the responses of the subsystems are superposed to form the response of the original large-order system.

The organization of the paper is as follows: In Section 2 the technical time-domain simulation problem will be discussed. Then, in Section 3 the flow chart and some important implementation details of `fastlsim` are presented. After describing the algorithm, Section 4 studies the simulation problems and solutions with various kinds of control loops. Then, in Section 5 simulation results are found for randomly generated systems and for a high-fidelity model of the Space Interferometry Mission (SIM) spacecraft. These are compared with those obtained by the standard MATLAB LTI systems simulator, `lsim`. Conclusions are summarized in Section 6.

2. TIME-DOMAIN SIMULATION PROBLEM

The time-domain simulation problem of a generic LTI system can be defined as follows: Given is the system in (2)

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t)\end{aligned} \tag{2}$$

[¶]For a diagonalizable matrix, the transformed matrix can be strictly diagonal but the result is usually complex, which is not very useful in practice. However, for a real diagonalizable matrix, the complex diagonal matrix can be converted into a real block diagonal one. Also, whenever the word 'diagonal' is used in this paper, it means real block diagonal, unless noted otherwise.

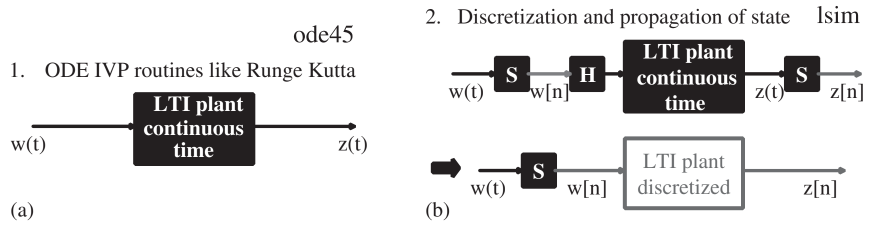

ode45
1. ODE IVP routines like Runge Kutta
LTI plant continuous time
w(t)
z(t)
(a)
2. Discretization and propagation of state
lsim
S
H
LTI plant continuous time
S
w(t)
w[n]
z(t)
z[n]
S
LTI plant discretized
w(t)
w[n]
z[n]
(b)

Figure 5. (a) Initial Value ODE solvers (e.g. `ode45`); and (b) state transition method (e.g. `lsim`). upper: true implementation, lower: discrete time equivalent. H=holder, S=sampler.

where $x(t)$ is the state, $u(t)$ is the input, $y(t)$ is the output and the matrices are of appropriate dimensions: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$. 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, and (2) The state transition method, see Figure 5.

Figure 5(a) shows the standard use of ODE solvers like Runge–Kutta. The meaning is straightforward: Given a system, external input and IC, the solver returns the response output. Although in a computer simulation $w(t)$ and $z(t)$ are not really continuous-time, these signals are still treated as continuous-time for the current purpose. The `ode45` solver in MATLAB belongs to the family of Runge–Kutta methods, in this case the Dormand–Prince [10] pair. Runge–Kutta is very versatile but it is more than needed for LTI systems. It computes five derivatives (number of matrix–vector products is doubled) for each time step and we will soon see that this is not necessary.

2.1. State propagation method

The second approach, which is the method proposed here, makes use of the known form of the solution to LTI systems, see Equation (3)

$$x(t) = \underbrace{e^{A(t-t_0)}x(t_0)}_{x_H} + \underbrace{\int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)\,d\tau}_{x_p} \tag{3}$$

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. This method includes discretizing a continuous time LTI system to its discrete-time counterpart using techniques like zero-order hold (ZOH) or first-order hold (FOH) and then computes the state transition through time using the discrete-time state equation, which is a recursive formula. This is equivalent to transforming the original problem (2) into its discretized version in (4), provided that a further assumption is made on $u(t)$ (e.g. ZOH).

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned} \tag{4}$$

where

$$A_d = \mathrm{e}^{AT}$$
$$B_d = \int_{KT}^{(K+1)T} \mathrm{e}^{A(KT+T-\tau)} B \, \mathrm{d}\tau = \int_0^T \mathrm{e}^{A\tau} B \, \mathrm{d}\tau$$

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

This method has the obvious advantage over `ode45` in that it requires only two matrix–vector products, namely the state transition matrix times the state vector and the input matrix times the input vector for each time step. In fact, MATLAB's `lsim` uses the method described here [11].

However, `lsim` is not without its own problems. Problems are encountered when simulating large-order systems like the full SIM model presented in Section 5. The first problem comes when one tries to discretize the original continuous time system. This requires computation of the matrix exponential e^{AT}. MATLAB calls the `c2d` function to do this (using ZOH, FOH or Tustin approximation). However, the computational time for `c2d` is not linear in n_s as it increases sharply with the order of the model. Also, `c2d` is general purpose and does not recognize or take advantage of special matrix structures like block diagonal matrices, which can accelerate the computational process. The second problem is even more severe. Simulators like `lsim` require a very large matrix to store the history of the state x_n. For instance, simulating the full SIM model with measured Magellan spacecraft reaction wheel assembly (RWA) disturbances, $w(t)$-six channels sampled at 4096 Hz for 210 s-requires $8 \times 2184 \times 210 \times 4096 \approx 15$ GB of memory, but 32-bit MATLAB on Windows only has 1.5 GB of memory available, including swap space. This means that standard simulators like `lsim` are still far from satisfactory for time-domain simulation of large LTI systems.

3. SIMULATION SCHEME: `fastlsim`

This section will address implementation issues of the proposed simulation scheme. Before the details are discussed, an overview of the algorithm is given in the flowchart of Figure 6. In this flow chart, the circles correspond to input or output data such as the original system A, B, C, D matrices, the external input $u(t)$ and computed output $y(t)$. Each block represents an operation or process with the corresponding MATLAB implementation labelled next to it.[‖]

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, Δt, 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 (4). The cost of this operation is $O(n_s^3)$ [12]. 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` for such an algorithm [13]), then the simulation might halt because of memory saturation. Solutions to these problems will be proposed in the following subsections.

[‖]The name of the corresponding m-file is in italic font.

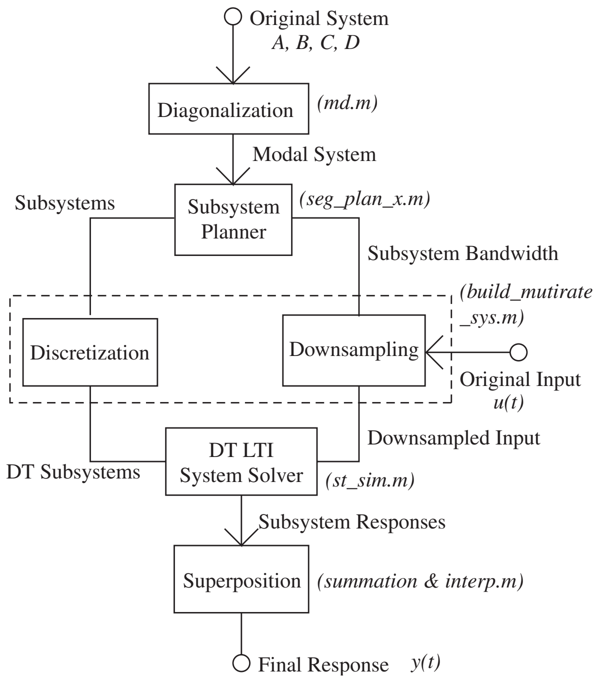


Figure 6. Flow chart of the `fastlsim` time-domain simulation algorithm.

3.1. Assumptions

Assumptions are vital in all scientific and engineering reasoning. In developing `fastlsim` and arguing its merits we make the following assumptions:

- The state matrix A is diagonizable, i.e. $A = V\bar{A}V^{-1}$, where $\bar{A}$ is block diagonal and real, and V is a matrix composed of columnwise eigenvectors.
- The sampling rate of the external input $u(t)$ is high enough.
- The analysis is carried out off-line.
- Performance measures are given as CPU time rather than FLOPS.
- Test machine specifications: Pentium 4, 1.8 GHz, 256 Mb RAM, Windows XP, IBM.

Since diagonalization is important in this method, the state matrix A is assumed to be diagonizable. Non-diagonizable matrices do exist but they are rare in practice. Some systems have repeated eigenvalues but this does not necessarily mean that they are defective since most of the time their eigenvectors still form a basis of the space of interest.

The highest frequency of the time-domain simulation is assumed to be the input sample rate, which is the reciprocal of the input time step. This is not necessarily the main limitation for further refinement of the method but is assumed here only for convenience. This defines the upper bound of the multiple sample rates to be used in `fastlsim`. Namely, it affects the way the subsystem planner is implemented.

Performance efficiency is given as CPU time. This is not the whole story since it is machine dependent. This is the only readily available tool for performance analysis in MATLAB 6 since a FLOPS count is no longer available with the incorporation of LAPACK in this version.

3.2. Diagonalization

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

$$A = S\Lambda S^{-1} \tag{5}$$

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, which is less expensive. Additionally, the corresponding subroutine of the presented simulation scheme (`md`) sorts the modes of the diagonalized system in ascending natural frequencies (i.e. the square root of the sum of squares of the real and imaginary parts of the eigenvalues). This diagonalization step is a necessary step to enable the following implementations:

- *Decoupling the dynamics and forming fictitious subsystems*. This can relieve the problem of memory saturation since the original large problem is divided into smaller subproblems. Also, this gives rise to the potential for parallel computation.
- *Applying multiple sampling rates*. The reason for this implementation is twofold. First, simulation can be facilitated by the application of multiple sampling rates. Secondly and more importantly, this can pave the way to solving multiple time scale dynamics problems, see Reich [14] for such a problem.
- *Exploiting the sparsity resulting from the diagonal structure of the A matrix*. The number of non-zero entries of an ordinary dense matrix $A \in \mathbb{R}^{n\times n}$ is n^2, but the number of non-zero entries in the diagonalized matrix is typically only $O(2n)$. An example is the baseline system ($n_s = 100$) in Figure 3(a), where there are $2.28 \times n_s$ non-zero entries, even though the system is not block diagonal. This sparsity is important in the matrix–vector product computation to be discussed later.

3.3. Subsystem planning

The subsystem planning subroutine is the key decision element in the algorithm. The objective of this function (`seg_plan_x`) is to form fictitious subsystems and to assign them appropriate sampling rates. The assumption in this subsection is that the plant is already block diagonal and that the modes are sorted (i.e. after the preceding diagonalization step). There are two considerations for subsystem planning.

The first issue is the size (in terms of number of state variables) of each subsystem. In terms of FLOPS, the size of the subsystems is not important because the number of FLOPS of simulating discrete-time systems (after diagonalization) depends linearly on the number of state variables. That is (see Reference [12], ignoring the FLOPS due to feedthrough)

$$\text{FLOPS} = 2 \times (2 + m + p) \times n_s \times n \tag{6}$$

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 time samples to be processed. Nevertheless,

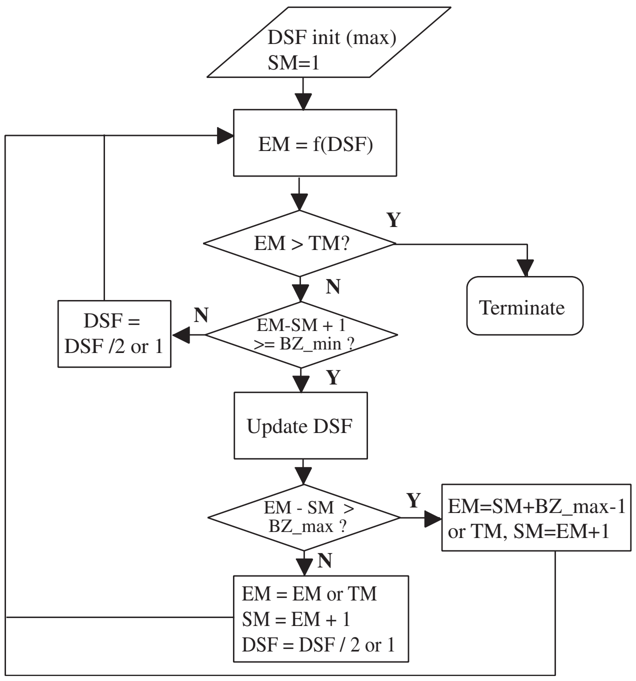


Figure 7. Flow chart of the subsystem planning subroutine.

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 associated with each subsystem. The minimum sampling rate is given by Nyquist's sampling theorem (e.g. see Reference [15]) 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, see also Equation (1). The issue of appropriate sampling rate selection has been extensively discussed by Franklin [16] and Åström [17] among others.

Taking into account of the aforementioned issues, the subsystem planning subroutine has the following features:

- It automatically chooses appropriate subsystems block sizes by considering a lower bound, upper bound and the effect of the number of input and output channels.
- It automatically suggests downsampling based on an estimate of the ratio between the high-frequency components and low-frequency components of the output.

The flow chart given in Figure 7 explains how the above features are achieved, where 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 and BZ is the currently chosen block size of each subsystem.

Given the initial downsampling rate (maximum allowable downsampling rate) and the first mode of the system, the block $f(\text{DSF})$ in Figure 7 computes the last mode of the subsystem

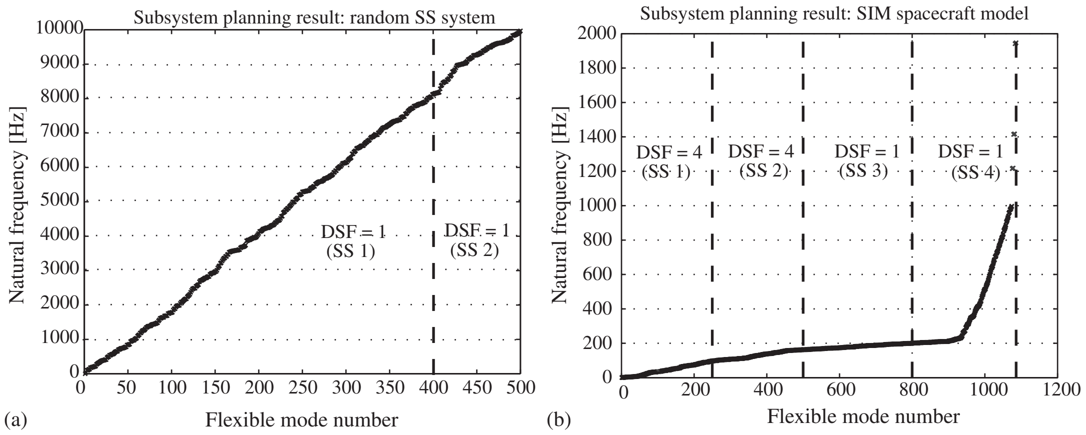


Figure 8. (a) Subsystem plan for random state space system with 500 modes and 10 kHz bandwidth; and (b) subsystems for SIM spacecraft model with 1092 modes and an input sampling rate of 4096 Hz.

so that it can be simulated with the current downsampling rate with an acceptable level of approximation, which is measured by the ratio between the power (or energy) of the outputs of the original and downsampled subsystem. The power (or energy) estimates are computed by fast Fourier transform (FFT). Since it is not straightforward to find in closed form the last mode of the subsystem with a given downsampling rate, what the subsystem planner does is to guess a subsystem size and then to evaluate its error. The guess is then sharpened iteratively using the bisection method. Once the last mode of the subsystem is determined, it is checked against the upper and lower bounds mentioned previously to adjust the subsystem size accordingly. Once a subsystem is determined, the DSF is decreased (i.e. increasing sampling frequency) and consideration of the next subsystem begins until all system modes have been assigned to a subsystem.

Planning results for two different systems are shown in Figure 8. In Figure 8(a) a random state space system was created with $n_s = 1000$ and $dr = 10\,000$ (same procedure as in Section 1.2). The modal frequencies increase linearly and `fastlsim` breaks the system into two subsystems (SS1: mode 1-400, SS2: mode 401-500), but downsampling is not used. In Figure 8(b) the subsystem planning results for the SIM flexible spacecraft model (see Section 5) are shown. The planner breaks the system into four chunks, two of which use a DSF = 4.

3.4. Discretization

As can be seen in Equation (4), the bottleneck of discretization is the matrix exponential e^{AT}. Fortunately, the diagonalization described in Section 3.2 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 Equation (4) is expressed as follows (see for example Reference [18]):

$$A_d = e^{AT} = I + AT + \frac{A^2T^2}{2} + \frac{A^3T^3}{3!} + \cdots \tag{7}$$

Table III. Computation time of eigenvalues (eig) and matrix exponential (expm) in seconds with test matrices of size n.

Size (n)	100	200	400	600	1000
eig	0.0310	0.2340	1.9850	6.4530	29.4530
expm	0.0310	0.3280	2.6250	9.0620	40.9530

By noticing the fact that

$$A^n T^n = \begin{bmatrix} A_1 T & 0 & \cdots \\ 0 & A_2 T & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}^n = \begin{bmatrix} A_1^n & 0 & \cdots \\ 0 & A_2^n & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} T^n \tag{8}$$

where $A_i \in \mathbb{R}^{2\times 2}$ $\forall i \in \{1, 2, \ldots\}$ and applying (8) to (7), the following equality holds:

$$\exp\left(\begin{bmatrix} A_1 T & 0 & \cdots \\ 0 & A_2 T & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}\right) = \begin{bmatrix} e^{A_1 T} & 0 & \cdots \\ 0 & e^{A_2 T} & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} \tag{9}$$

Equation (9) is the basis of the $O(n_s)$ discretization scheme implemented here. As a result, a generic procedure for the fast discretization can be given as

```
for $i = 1$ to total number of subsystems
  index=location of subsystem ;
  DT_system(index)=c2d(subsystem) ;
end
```

Here the size of each subsystem is not fixed and a theoretical optimal block size can be found. Nevertheless, the optimality is not an important issue since the difference between optimal and suboptimal strategies is not obvious here (this is in contrast to the case of the fast Lyapunov solver discussed in de Weck *et al.* [2]). A comparison of the time required for computing the matrix exponential between a 1000×1000 dense matrix (27.1 s) and a block diagonal matrix (0.031 s) of the same dimensions reveals that the diagonal matrix exponential is 874 times faster, yielding the same result.

Nevertheless, the advantage of fast discretization comes with the expense of the diagonalization step, which is also $O(n^3)$. Fortunately, the computation of eigenvalues is usually more efficient than that of the matrix exponential, as Table III shows.

3.5. Downsampling

Downsampling is required whenever the assigned sampling rate of a particular simulation is lower than that of the input. There are two ways to implement downsampling (Figure 9).

The first way is to low-pass filter the input signal, thus removing its high-frequency components. The straightforward way to achieve the goal is to low-pass filter the signal and then

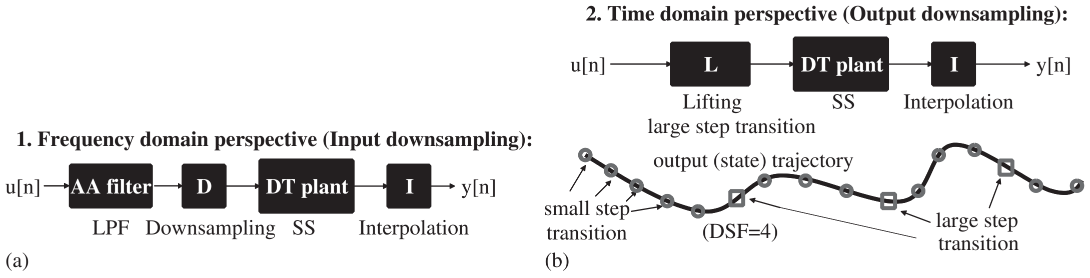


Figure 9. (a) Input downsampling; and (b) output downsampling.

downsample it (see for example Reference [15]). That is

$$x_d[n] = x[Mn] \tag{10}$$

where $x[n]$ has been low-pass filtered and M is the DSF. This is essentially downsampling the input to the system.

The second way is to reformulate the state transition formula so that all input instances affect the state transition, but some intermediate states (not at major time step) are not explicitly computed, thus saving computational effort. The time-domain downsampling scheme borrows the idea from the more general scheme of lifting [27]. Recall the state transition formula (or the state equations for the discretized system)

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned} \tag{11}$$

The first equation in (11) is certainly satisfied at the $n+1$ instant

$$x[n+2] = A_d x[n+1] + B_d u[n+1] \tag{12}$$

If (11) is substituted into in (12), then the following results:

$$\begin{aligned} x[n+2] &= A_d\{A_d x[n] + B_d u[n]\} + B_d u[n+1] \\ &= A_d^2 x[n] + A_d B_d u[n] + B_d u[n+1] \end{aligned} \tag{13}$$

Equation (13) can be generalized for N steps,

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

If the following new matrices are defined

$$\underline{u}[n] = \left[u[n] \ \cdots \ u[n+N-1]\right]^{\mathrm{T}}$$

$$\underline{A_d} = A_d^N$$

$$
\begin{aligned}
\underline{B_d} &= \begin{bmatrix} A_d^{N-1} B_d & \cdots & B_d \end{bmatrix} \\
\underline{C} &= C \\
\underline{D} &= \begin{bmatrix} D & 0 & \cdots & 0 \end{bmatrix}
\end{aligned}
$$

then the N-step propagation version of (11) is

$$
\begin{aligned}
x[n+N] &= \underline{A_d} x[n] + \underline{B_d} \underline{u}[n] \\
y[n] &= \underline{C} x[n] + \underline{D} \underline{u}[n]
\end{aligned}
\tag{14}
$$

By employing a long time step state transition, the calculation of the unwanted intermediate state and output 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 (15) and those of the second method (16):

$$
\text{FLOPS} = 2 \times \frac{(2+m+p) \times n_s \times n}{\text{DSF}} \tag{15}
$$

$$
\text{FLOPS} = 2 \times \frac{(2+\text{DSF} \times m+p) \times n_s \times n}{\text{DSF}} \tag{16}
$$

Here $\text{DSF} \geqslant 1$ is the downsampling factor. The meanings of other parameters in (15) and (16) are referred back to Equation (6). In conclusion, the second downsampling method is less efficient, but more conservative from an error management perspective and is the one used in the `fastlsim` algorithm.

3.6. 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 and the original ODE problem now becomes a much simpler problem of matrix–vector multiplication (cf. Equation (4)). 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. The required features of the simulator are summarized as follows:

- The simulator must be memory conscious. It cannot request any amount of memory (in bytes) proportional to $n_s n$ or more.
- The simulator must be able to recognize the zero patterns of the A matrix, otherwise, the advantage of the sparsity will be lost and the computational effort estimate in Equation (6) will not be achieved.

Interpolation is needed whenever the output is downsampled. It should be pointed out that the choice of interpolation scheme is a tradeoff between computation effort and accuracy. More details can be found in Sou [12].

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 which allows for superposition of the subsystem responses.

4. CLOSED LOOP SYSTEMS ISSUES

In this section practical issues concerning the implementation of the `fastlsim` algorithm are addressed. The main problem is due to closing a feedback loop (e.g. attitude control systems for a satellite).

4.1. Simulation with feedback loops

The block diagram of the problem with feedback loops is given in Figure 2. Suppose the open loop system is in modal form (i.e. the A matrix is block diagonal) and the state space form is**

$$\left[\begin{array}{c|cc} A & B_w & B_u \\ \hline C_z & D_{zw} & D_{zu} \\ C_y & D_{yw} & 0 \end{array}\right] \tag{17}$$

The feedback controller has the following state space realization:

$$\begin{bmatrix} A_k & B_k \\ C_k & D_k \end{bmatrix} \tag{18}$$

where subscripts w and u denote 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. The closed loop system has the state space form

$$\left[\begin{array}{cc|c} A + B_u D_k C_y & B_u C_k & B_w + B_u D_k D_{yw} \\ B_k C_y & A_k & B_k D_{yw} \\ \hline C_z + D_{zu} D_k C_y & D_{zu} C_k & D_{zw} + D_{zu} D_k D_{yw} \end{array}\right] \tag{19}$$

The off-diagonal blocks, $B_u C_k$ and $B_k C_y$ in the 'A' matrix of (19) are not expected to be zero, otherwise there will be no control effect at all. Now the problem is: Even if the open loop system is in modal form (i.e. the A matrix is block diagonal), the closed loop system will not be so because of the dynamics coupling (off diagonal terms in 'A' in (19)). Another problem arises if the feedback controller is a discrete-time system†† and this causes the closed loop system to be hybrid.‡‡ For the problem of dynamics coupling, two solutions are proposed:

Rediagonalization by eigenvalue decomposition: An eigenvalue decomposition is applied to the system in Equation (19). This is the most straightforward way but the computation can be expensive if the systems considered are large-order, because of the eigenvalue problem involved.

Forced decoupling: This is a heuristic method in that some of the entries of C_y in Equation (19) are set to zero. In other words, some of the measurements are regarded as

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

††In practical implementations, controllers are usually digital, which implies that the signals, as well time instants, are discrete. The discrete-time assumption here is merely for ease of analysis.

‡‡Here the term 'hybrid' is used in the very specific sense that the system contains both continuous-time and discrete-time states. There are no discrete states involved.

insensitive to some of the state variables. Suppose for simplicity that all D matrices are zero and the state variables are reordered in such a way that the following equalities hold (A is assumed to be block diagonal and C_y is partitioned into two blocks):

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$$

$$B_u = \begin{bmatrix} B_{1u} \\ B_{2u} \end{bmatrix}$$

$$B_w = \begin{bmatrix} B_{1w} \\ B_{2w} \end{bmatrix}$$

$$C_y = \begin{bmatrix} 0 & C_{2y} \end{bmatrix}$$

$$C_z = \begin{bmatrix} C_{1z} & C_{2z} \end{bmatrix}$$

Then the state space representation of the closed loop system (19) is as follows:

$$\left[\begin{array}{ccc|c} A_1 & 0 & B_{1u}C_k & B_{1w} \\ 0 & A_2 & B_{2u}C_k & B_{2w} \\ 0 & B_kC_{2y} & A_k & 0 \\ \hline C_{1z} & C_{2z} & 0 & 0 \end{array}\right] \tag{20}$$

It can be verified that the system in (20) can be decomposed into two subsystems. The first subsystem includes controller dynamics and is subject to disturbance input only

$$\left[\begin{array}{cc|c} A_2 & B_{2u}C_k & B_{2w} \\ B_kC_{2y} & A_k & 0 \\ \hline C_{2z} & 0 & 0 \end{array}\right] \tag{21}$$

and the second subsystem evolves in time with disturbance input and control input that is determined by solving the first subsystem (as the output signal of the controller)

$$\left[\begin{array}{c|cc} A_1 & B_{1u} & B_{1w} \\ \hline C_{1z} & 0 & 0 \end{array}\right] \tag{22}$$

It can be seen that if the dimension of A_2 in (21) is much smaller than that of A_1 in (22) and if A_1 is block diagonal, then the bottleneck of diagonalization can be avoided. The justification of this method hinges upon the ability to find the state variables that are insensitive to sensor measurements and the relative significance of the contributions of the ignored measurements to the total measurements. The determination of the 'important' state variables can be quite case specific. For example, in the study of a satellite structure with an attitude control system (ACS), if the measurements are attitude angles, then it is natural that the rigid body modes are far more important than other flexible modes (Section 5). In order to quantify the error induced by the forced decoupling method, it is possible to compute the ratio

$$E = \frac{\sigma_1}{\sigma_2} \tag{23}$$

Table IV. E: % of RMS attitude angles contributed by flexible modes relative to rigid body subsystem.

θ_1	θ_2	θ_3
0.1021	0.2201	0.1950

where σ_1 and σ_2 are the open (feedback control) loop RMS values of the contributions of the unimportant and important dynamics to the measurement. The computation of the ratio E in (23) can be very efficient if the open loop system is already in modal form.

If E is smaller than some tolerance, then the block of the C matrix corresponding the unimportant dynamics is small and the forced decoupling heuristic is promising. Otherwise, rediagonalization by eigenvalue decomposition must be applied. As an example, consider the 2184 state variable SIM model (Section 5) with three unstable (or marginally stable) rotational rigid body modes and assume the attitude angles (proportional to rigid body mode angles, together with some additional contributions from other flexible modes) are measured directly. The numeric values of E in (23) in this example are summarized in Table IV. The results in Table IV show that E is small if the forced decoupling heuristic is used.

To verify the prediction, the actual results by the two methods are computed: The RMS values of the performance outputs obtained by eigenvalue decomposition and by the forced decoupling heuristic are 1.8275×10^{-5} and 1.8276×10^{-5}, respectively. The difference is 1.8135×10^{-9}, which amounts to about 0.0099% of the performance given by eigenvalue decomposition method (chosen as a reference here). In conclusion, the forced decoupling heuristic is not exact but can be fairly accurate if properly applied.

The problem of 'hybrid' closed loop systems is addressed in Appendix B. The second frequently encountered problem is due to a cascade connection between the plant and some other filters (e.g. noise shaping filter and/or controller post-filter). Solutions to avoid rediagonalization with cascading filters are provided in Sou [12].

5. SIMULATION RESULTS

In this section, `fastlsim` is applied to example problems to show its potential value. We first apply the algorithm to randomly generated state space systems and try to quantify the computational benefit. Next, we show that the scheme enables time-domain simulation for larger order systems, such as the 2184 state SIM spacecraft model, that could not previously be solved on single PC-class computers.

5.1. Simulation of random systems

The first example computes the RMS values of the outputs of randomly generated stable SISO systems of different dimensions (n_s) driven by randomly generated input signals. As in Section 1 three methods are compared: time-domain simulation, frequency-domain analysis and Lyapunov analysis. 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 V.

Table V. Comparison between time-domain, frequency-domain and Lyapunov methods for randomly generated systems of different size.

n_s	Results	`freq`	`lyap`	`lsim`	`fastlsim`
50	T_{CPU}[s]	0.2960	0.0780	1.3750	0.6880
	σ	1.714E − 6	1.713E − 6	1.722E − 6	1.722E − 6
100	T_{CPU}[s]	0.6410	0.2340	4.5310	1.0150
	σ	1.167E − 5	1.159E − 5	1.187E − 5	1.187E − 5
200	T_{CPU}[s]	2.0000	1.375	71.078	2.3900
	σ	1.516E − 5	1.448E − 5	1.463E − 5	1.463E − 5
500	T_{CPU}[s]	15.3280	13.859	N/A	15.2030
	σ	3.006E − 5	3.002E − 5	N/A	3.061E − 5
1000	T_{CPU}[s]	104.25	98.562	N/A	96.0320
	σ	5.452E − 5	5.184E − 5	N/A	5.228E − 5
1500	T_{CPU}[s]	425.3	393.3	N/A	382.7
	σ	1.346E − 5	1.325E − 5	N/A	1.334E − 5
2000	T_{CPU}[s]	1046.4	949.3	N/A	934.4
	σ	2.869E − 5	2.714E − 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) for each computation and σ is the RMS value of each output (the actual units are not of interest here). While `freq` denotes the frequency-domain method, `lyap` denotes the Lyapunov method and `lsim` is the standard time-domain simulation method provided by MATLAB. The time-domain simulation algorithm presented in Section 3 is designated as `fastlsim`. Note that `freq` and `lyap` are the fast implementations of the frequency-domain method and Lyapunov method, respectively, see de Weck *et al.* [2]. Note also that the time shown for `fastlsim` includes the time to diagonalize.

Since new data are generated in each case with a different n_s, the RMS values of different n_s cases differ accordingly and should not be 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 V, 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 times for the frequency-domain and time-domain methods vary with the number of samples evaluated. Nevertheless, it is this computation that provides the additional information that the Lyapunov method does not provide (i.e. time history and power spectral density). The reason why the results from `lsim` are unavailable (N/A) is that the computer ran out of memory, which proves that `lsim` is not suited for large-order systems simulation. The above example shows that `fastlsim` can achieve efficiency similar to the fast implementations of other methods (frequency-domain and Lyapunov methods) with acceptable accuracy when computing RMS values.

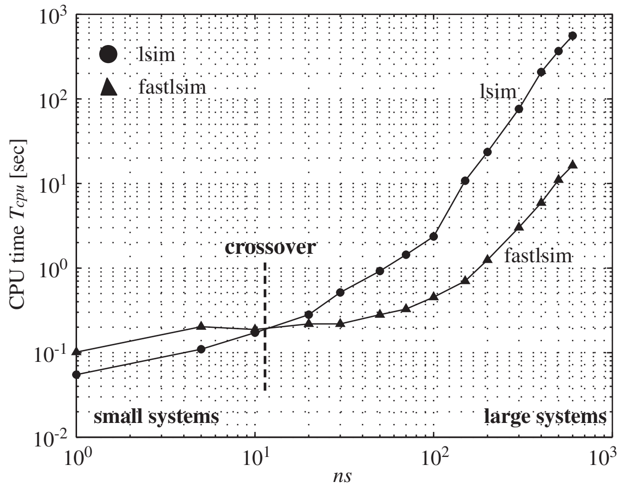
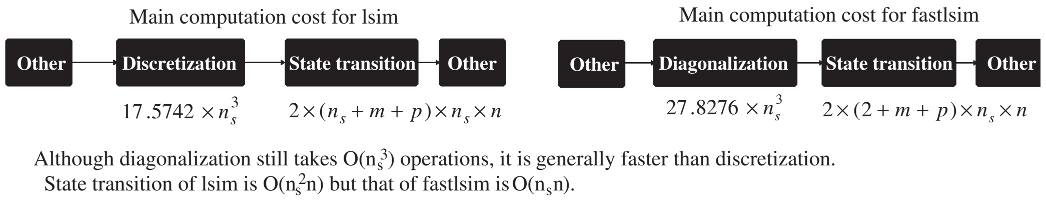


Figure 10. Plot of CPU times for `lsim` and `fastlsim` as a function of n_s.



Main computation cost for lsim
Other
Discretization
State transition
Other
$17.5742 \times n_s^3$
$2\times(n_s+m+p)\times n_s\times n$
Main computation cost for fastlsim
Other
Diagonalization
State transition
Other
$27.8276 \times n_s^3$
$2\times(2+m+p)\times n_s\times n$
Although diagonalization still takes $O(n_s^3)$ operations, it is generally faster than discretization.
State transition of lsim is $O(n_s^2 n)$ but that of fastlsim is $O(n_s n)$.

Figure 11. Computational cost budget for `lsim` and `fastlsim`.

Computing output RMS values does not exemplify the true advantage of `fastlsim`. A time-domain simulation scheme should be compared with another time-domain simulation scheme. Therefore the MATLAB simulator, `lsim`, 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` and `fastlsim`. The computation times for each simulation are given in Figure 10.

A savings factor $T_{\mathrm{lsim}}/T_{\mathrm{fastlsim}}$ of over 50 was shown before `lsim` ran out of memory. Crossover occurs at around $n_s = 12$ state variables, which means that even small to mid-sized system are handled effectively by `fastlsim`.

Figure 11 gives a rudimentary operations count for `lsim` and `fastlsim`. All $O(n_s^3)$ results are found empirically and the state transition operation count is straightforward (just count the matrix–vector products and consider sparsity). It turns out that $O(n_s^3)$ computation is inevitable, but computing eigenvalues is far more efficient than computing matrix exponentials (there is a much greater body of research on the former problem). Also, state transition for `fastlsim` is much faster than `lsim`, considering the value of n (number of time simulation samples).

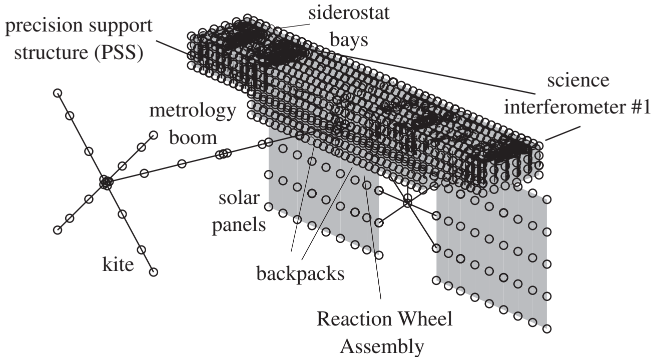


Figure 12. Finite element model of SIM v2.2; approx. 20 000 DOF.

Table VI. 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)

An empirical fit of the equation $T_{\text{CPU}} = \alpha \times n_s^{\beta}$ reveals coefficients of $\alpha = 10^{-5}$ and $\beta = 2.82$ for `lsim` and coefficients of $\alpha = 5.2 \times 10^{-6}$ and $\beta = 2.35$ for `fastlsim`. This again confirms that we have achieved our goal of developing a fast time-domain simulator which scales closer to $O(n_s^2)$ than $O(n_s^3)$ in the region of interest of large LTI systems (see also Figure 1). The `fastlsim` method is found to be highly accurate with relative averaged point to point errors below 0.5%; see Sou [12] for details on error analysis.

5.2. Application to SIM spacecraft model

The remaining example is concerned with simulation and control tuning for the Space Interferometry Mission (SIM) spacecraft [19], see Figure 12 for its finite element model. This task was enabled by `fastlsim` and was previously not feasible due to memory saturation.

The SIM model presents significant challenges for time-domain simulation because of its high dimensionality (2184 state variables) and wide dynamic range ($\omega_{\min}/\omega_{\max} \approx 4700$).[§§] Due to its scientific mission, stringent design requirements are imposed on the performance outputs of the SIM model. For example, see Table VI for some requirements from Miller *et al.* [20] and Laskin [21] for the optical pathlength difference (OPD) and wavefront tilt (WFT) of science interferometer #1.

[§§] $\omega_{\min}$ is the minimum flexible mode natural frequency. The zero natural frequency of the rigid body modes is not counted.

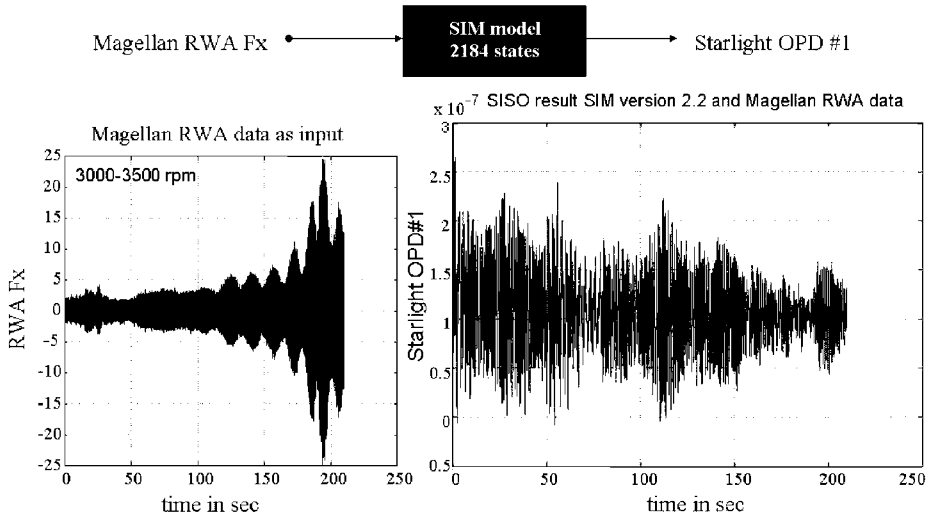


Figure 13. Time-domain simulation of SIM v2.2 (SISO case).

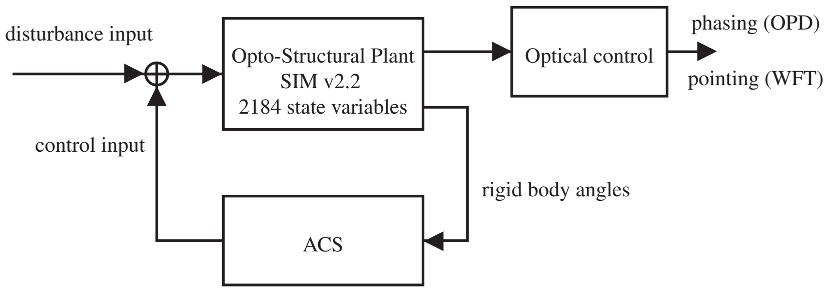


Figure 14. Appended LTI system dynamics of SIM.

The main mechanical disturbances are generated by the reaction wheel assembly (RWA) with three channels of force and three channels of torque. These vibrations travel through the flexible structure to adversely impact the performance metrics shown in Table VI.

Figure 13 shows time-domain simulation results obtained by `fastlsim` for an open loop SIM SISO model (RWA F_x force to OPD #1). The disturbances $w(t)$ are obtained based on laboratory measurements of the Magellan spacecraft reaction wheels, sampled at 4096 Hz. The results presented in Figure 13 took 46.63 s to compute. Previous attempts using `lsim` or `ode45` were unsuccessful due to the aforementioned memory saturation problem. The full MIMO time-domain simulation with six input channels and six output channels took 217.63 s to compute.

In order for the SIM system to work properly, two control systems are needed. One is the ACS and the other is the optical control system. The overall system configuration is given in Figure 14. The ACS in Figure 14 stabilizes the open loop unstable rigid body modes of

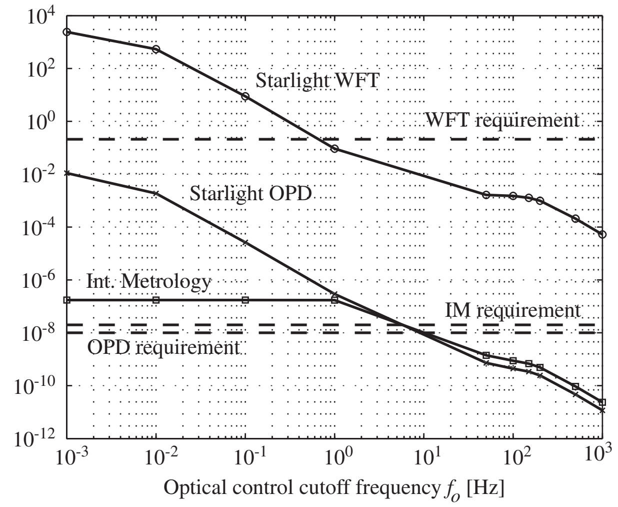


Figure 15. RMS and RSS of the performance outputs. Circle: Starlight OPD #1. Asterisk: Internal Metrology OPD #1. Square: Starlight WFT #1.

the SIM model. It can be designed by classical methods like PID, lead-lag or modern control techniques such as LQG. The optical control here is modelled as a second-order high-pass filter and the transfer function of one channel is

$$K_o(s) = \frac{s^2}{s^2 + 2\zeta_o \omega_o s + \omega_o^2} \tag{24}$$

where ζ_o is the damping ratio of the controller which is set to 0.707 and ω_o is the corner frequency, which is treated as a design parameter. This allows conducting a parameter study of optical controller corner frequency ω_o [rad/s] (or f_o [Hz]). The system consists of the open loop SIM model (2184 state variables), an ACS designed by the standard LQG approach (e.g. Reference [22]) and the optical controller as given in (24).

The ACS loop is closed by a rediagonalization by eigenvalue decomposition and the optical control path is closed by the method described in Sou [12]. There are six input channels (three forces and three torques), which are driven by the six channels of Magellan reaction wheel assembly disturbance data (see Reference [23]). The outputs are starlight optical path difference (OPD), internal metrology (IM) and starlight WFT. In the simulation runs, different closed loop systems with different optical controller corner frequencies (f_o) are formed and the RMS values of the performance outputs are recorded. The result is shown in Figure 15.

The result in Figure 15 is consistent with the intuition that the more optical control is applied, the better the performances are. Nevertheless, it can be a problem of cost and implementation if f_o is too high. The results are obtained efficiently with `fastlsim` and reveal that minimum optical control bandwidths of 1 Hz for WFT and 10 Hz for OPD and IM are necessary in order for SIM to meet its scientific requirements.

6. CONCLUSIONS

In this paper, a new time-domain simulation scheme, `fastlsim`, based on block diagonalization is presented. 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. Problems with feedback and feed-forward controllers are discussed and the corresponding solutions (e.g. forced decoupling and rediagonalization without using an iterative eigenvalue solver) are proposed. It is shown, using randomly generated stable state space systems, that `fastlsim` outperforms existing algorithms such as `lsim` by factors up to 50, while avoiding memory saturation. Surprisingly, this scheme can achieve the same efficiency as that of fast implementations of frequency-domain and Lyapunov methods. Real world simulations of complex systems, such as the $\approx$2200-state closed loop SIM spacecraft, have been enabled and demonstrated.

Future work includes further refinement of error analysis and error control, potential coupling of `fastlsim` with model reduction, better ways of incorporating cascading filters and controllers as well the extension of the algorithm on parallel processors and weakly non-linear systems.

APPENDIX A: ANALYSIS METHODS TO COMPUTE SYSTEM RESPONSE σ_z

The starting point is a state space system of the form shown in Figure 2. This system can be open loop or closed loop (i.e. non-zero A_k, B_k, C_k, D_k):

$$\begin{aligned} \dot{x} &= A_{zw}x(t) + B_{zw}w(t) \\ z(t) &= C_{zw}x(t) + D_{zw}w(t) \end{aligned} \tag{A1}$$

Once such an 'integrated' model is available, one may want to assess the performance of the system when its model is subjected to disturbances, $w(t)$. The RMS $\mathscr{H}_2$-performance metric according to Zhou [24] is given as follows:

$$\sigma_z = E[z^{\mathrm{T}}z]^{1/2} = \left[\frac{1}{T}\int_0^T z^2(t)\,\mathrm{d}t\right]^{1/2} \text{RMS} \tag{A2}$$

The RMS metric is typically used to describe the 'on-average' performance of a system. Three analysis approaches are discussed below that are helpful in obtaining estimates of σ_z.

A.1. Time-domain simulation (time)

A linear time-invariant system is given in the form of Equation (A1), where x is the state vector. Equivalently, the system can be described in the frequency-domain by the transfer function matrix

$$G_{zw}(\omega) = C_{zw}\left[j\omega I - A_{zw}\right]^{-1} B_{zw} + D_{zw} \tag{A3}$$

When measured or synthetically generated time histories of the disturbances $w(t)$ exist, they can be used for time integration of the state space equations (A1). Once the initial condition on the state vector, $x\,(t=0)$, is specified, numerical integration of (A1) can then be performed to obtain estimates of the performance time histories $z(t)$. The standard difference method technique approximates the continuous first-order equation (A1) with a difference equation such as

$$\begin{aligned} \frac{(x)_{n+1} - (x)_n}{\Delta t} &= A_{zw}(x)_n + B_{zw} w_n \\ z_n &= C_{zw}(x)_n + D_{zw} w_n \end{aligned} \tag{A4}$$

In the simplest approach the state vector $(x)_{n+1}$ at the $n+1$th time step can be found by the forward Euler method as

$$(x)_{n+1} = \left[\Delta t A_{zw} + I\right](x)_n + \Delta t B_{zw} w_n \tag{A5}$$

This integration method is simple but can diverge easily when $\Delta t \geqslant \Delta t_{\text{crit}}$. Higher order ordinary differential equation solvers such as `ode45` according to Dormand and Prince [10] give better results, but can be computationally more expensive. An advantage of the time-domain approach is that transient effects can be observed.

A.2. Frequency-domain analysis (freq)

For linear systems in the time domain, the output can be expressed as a convolution of the input with the impulse-response function of the system. In the frequency domain (i.e. Laplace domain), the output is then equal to the input multiplied by the transfer function (matrix). The disturbance spectral density matrix $S_{ww}(\omega)$ can be measured experimentally or obtained from a shaping pre-filter as $S_{ww}(\omega) = G_d(\omega)G_d^H(\omega)$. The performance spectral density matrix S_{zz} can be obtained from Reference [25] as

$$S_{zz}(\omega) = G_{zw}(\omega)S_{ww}(\omega)G_{zw}^H(\omega) \tag{A6}$$

where S_{ww} is the disturbance spectral density matrix discussed above and G_{zw} is the open or closed loop plant transfer function matrix from (A3). The covariance matrix of the performances Σ_z (for zero-mean processes) is obtained as

$$\Sigma_z = \frac{1}{2\pi}\int_{-\infty}^{+\infty} S_{zz}(\omega)\,\mathrm{d}\omega = \int_{-\infty}^{+\infty} S_{zz}(f)\,\mathrm{d}f \tag{A7}$$

The variance of the performance is therefore given by

$$\begin{aligned} \sigma_z^2 = \left[\Sigma_z\right] &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} \left[S_{zz}(\omega)\right]\mathrm{d}\omega \\ &= \int_{-\infty}^{+\infty} \left[S_{zz}(f)\right]\mathrm{d}f = 2\int_{0}^{+\infty} \left[S_{zz}(f)\right]\mathrm{d}f \end{aligned} \tag{A8}$$

Taking the square root of σ_z^2 produces the RMS value. It is important to specify whether a PSD is one or two sided and given in Hz or rad/s [25]. In practice the upper and lower frequency integration limits are $f_{\min}$ and $f_{\max}$, respectively.

$$\sigma_z^2 \approx 2 \int_{f_{\min}}^{f_{\max}} \left[S_{zz}(f) \right] \mathrm{d}f \tag{A9}$$

It is important to ensure that the frequency range that contributes most to the RMS value is sufficiently captured within these limits. One way to verify this is by computing the cumulative RMS function $\sigma_{z,c}(f_o)$ as

$$\sigma_{z,c}(f_o) = \left[2 \int_{f_{\min}}^{f_o} \left[S_{zz}(f) \right] \mathrm{d}f \right]^{1/2} \tag{A10}$$

where $f_o \in [f_{\min} \dots f_{\max}]$. If most of the energy lies in this range, then $\sigma_{z,c}(f_{\max})$ should be very close to the true value of σ_z. The method requires high-frequency resolution near lightly damped modes in order to arrive at correct RMS values. Also the frequency-domain method is not well suited to assess the transient performance of a linear time-invariant system.

A.3. Lyapunov analysis (lyap)

The third type of disturbance analysis can be conducted if the disturbances w are modelled as the outputs of a pre-shaping filter (A_d, B_d, C_d, D_d) as shown in Figure 2. In order to keep the disturbance w from having infinite energy, the feedthrough matrix D_d should be zero. If the system is asymptotically stable, the state covariance matrix obeys the Lyapunov equation [26].

$$A_{zd}\Sigma_x + \Sigma_x A_{zd}^{\mathrm{T}} + B_{zd} B_{zd}^{\mathrm{T}} = \dot{\Sigma}_x \tag{A11}$$

In order to do time integration of the above dynamics, the initial state covariance, Σ_{x_o}, would have to be specified. Since the white noise disturbance processes, d, are assumed to be stationary, the statistics of the state vector are also stationary and $\dot{\Sigma}_x = 0$. One may then solve the steady-state Lyapunov equation of order n_s for the state covariance matrix Σ_x of the system.

$$A_{zd}\Sigma_x + \Sigma_x A_{zd}^{\mathrm{T}} + B_{zd} B_{zd}^{\mathrm{T}} = 0 \tag{A12}$$

One can pre- and post-multiply with the entire C_{zd} matrix to obtain the performance covariance matrix Σ_z.

$$\Sigma_z = E\left[zz^{\mathrm{T}}\right] = E\left[C_{zd} x x^{\mathrm{T}} C_{zd}^{\mathrm{T}}\right] = C_{zd} E\left[x x^{\mathrm{T}}\right] C_{zd}^{\mathrm{T}} = C_{zd} \Sigma_x C_{zd}^{\mathrm{T}} \tag{A13}$$

The variances of the individual performances (RMS squared) are then contained on the diagonal of Σ_z, where Σ_z is of the form

$$\Sigma_z = \begin{bmatrix} \sigma_{z_1}^2 & \sigma_{z_1 z_2} & \cdots & \sigma_{z_1 z_n} \\ \sigma_{z_2 z_1} & \sigma_{z_2}^2 & \cdots & \sigma_{z_2 z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{z_n z_1} & \sigma_{z_n z_2} & \cdots & \sigma_{z_n}^2 \end{bmatrix} \tag{A14}$$

Thus, the Lyapunov method provides a relatively direct way of arriving at the RMS estimates (in the sense of statistical steady state) by solving one matrix equation (A12) and computing a matrix triple product (A13). The main drawback of the Lyapunov approach is that it does not provide insight into the frequency content of the outputs. The main advantage is that the answers provided are immune to the frequency resolution (Δf) and time step (Δt) issues associated with the other two methods.

APPENDIX B: DEALING WITH HYBRID CLOSED LOOP SYSTEMS

We suggest two options to deal with hybrid closed loop systems:

Continuous-time approximation: If the sampling rate of the control is high enough, then the digital controller can actually be approximated by a continuous-time system using techniques like ZOH or bilinear (Tustin) transform. The accuracy of this approximation can be found in any common digital (or computer) control literature.

Lifting: This is a useful approach to deal with sampled-data control systems analysis problems. For example, see Chen [27] and Yamamoto [28]. Suppose a discrete-time signal $v[n]$ is defined as

$$v = \{v[0], v[1], v[2], \ldots\}$$

The lifted version of the signal $\underline{v}$ can be expressed as

$$\underline{v} = \left\{ \begin{bmatrix} v[0] \\ v[1] \\ \vdots \\ v[n-1] \end{bmatrix}, \begin{bmatrix} v[n] \\ v[n+1] \\ \vdots \\ v[2n-1] \end{bmatrix}, \ldots \right\}$$

where $n \in \mathbb{Z}$. If an LTI system is represented as

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

then its lifted version is defined such that the original inputs and outputs signals are lifted. That is

$$\left[\begin{array}{c|cccc} A^n & A^{n-1}B & A^{n-2}B & \cdots & B \\ \hline C & D & 0 & \cdots & 0 \\ CA & CB & D & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{n-1} & CA^{n-2}B & CA^{n-3}B & \cdots & D \end{array}\right] \tag{B1}$$

The lifting procedure in (B1) basically reduces sampling rate at the expense of an increase in input and output dimensions. Equivalently, this procedure can be viewed as one of the applications of the state augmentation technique. The main application of this method in the paper is to convert a multiple sampling rates[¶¶] system into a single rate (slow rate) LTI system without losing the effect of fast dynamics. The drawback of this method is the high resulting dimensionality. Nevertheless, this method can work well in conjunction with the forced decoupling method discussed previously if the subsystem coupled with the controller is of low dimension.

ACKNOWLEDGEMENTS

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. The `fastlsim` simulation package, including source code, a user manual and examples are available from the corresponding author.

REFERENCES

1. Gutierrez H. Performance assessment and enhancement of precision controlled structures during conceptual design. *Ph.D. Thesis*, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 1999.
2. de Weck O, Uebelhart S, Gutierrez H, Miller D. Performance and sensitivity analysis for large order linear time-invariant systems. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, September, 2002, Atlanta, GA, USA.
3. de Weck O. Multivariable isoperformance methodology for precision opto-mechanical systems. *Ph.D. Thesis*, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, September, 2001.
4. Mallory G, Miller D. Increasing the numerical robustness of balanced model reduction. *Journal of Guidance*, *Control*, *and Dynamics* 2002; **25**(3):596–598 (Engineering notes).
5. Willcox K, Peraire J. Balanced model reduction via the proper orthogonal decomposition. *15th AIAA Computational Fluid Dynamics Conference*, June, 2001, Anaheim, CA, USA.
6. Beran P, Silva W. Reduced-order modelling: new approaches for computational physics. *39th Aerospace Sciences Meeting and Exhibit*, January, 2001, Reno, NV, USA.
7. Giesing J, Barthelemy J-F. A summary of industry MDO applications and needs. *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Optimization and Analysis*, *AIAA Paper No. 98-4737*, September, 1998, St. Louis, MO, USA.
8. Sobieszczanski-Sobieski J, Haftka R. Multidisciplinary aerospace design optimization—survey of recent developments. *AIAA 34th Aerospace Sciences Meeting and Exhibit*, January, 1996, Reno, NV, USA.
9. Anderson M, Mason W. An MDO approach to control-configured-vehicle design. *The 6th AIAA*, *NASA*, *and ISSMO*, *Symposium on Multidisciplinary Analysis and Optimization*, September, 1996, Bellevue, WA, USA.
10. Dormand JR, Prince PJ. A family of embedded Runge–Kutta formulae. *Journal of Computational and Applied Mathematics* 1980; **6**:19–26.
11. The Mathworks Inc. *MATLAB*, Control Systems Toolbox, Version 6.5.1, Release 13, August 2003.
12. Sou KC. Fast time domain simulation for large order hybrid systems. *Master Thesis*, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, May, 2002.
13. Andrew G. *Control System Toolbox User's Guide*. The MathWorks, Inc.: Natick, MA, 1996.
14. Reich S. Multiple time scales in classical and quantum classical molecular dynamics. *Journal of Computational Physics* 1997; **151**(1):49–73.
15. Oppenheim A, Schafer R, Buck J. *Discrete-time Signal Processing* (2nd edn). Prentice-Hall: Englewood Cliffs, NJ, 1999.
16. Franklin G, Powell J, Workman M. *Digital Control of Dynamic Systems* (2nd edn). Addison-Wesley: Reading, MA, 1990.

[¶¶] Sometimes the sampling rate of the plant is much higher than that of the controller, in order to represent the plant dynamics accurately, see Chen [27] for more details. Note also that a multiple sampling rate system is time-variant.

17. Åström K, Wittenmark B. *Computer-Controlled Systems*: *Theory and Design*. Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1997.
18. Chen C. *Linear System Theory and Design*. Oxford University Press: Oxford, 1999.
19. NASA Jet Propulsion Laboratory. Internet link: http://sim.jpl.nasa.gov/.
20. Miller D, de Weck O, Uebelhart S. Integrated dynamics and controls modeling for the space interferometry mission (SIM). *IEEE Aerospace Conference*, March, 2001, Big Sky, MT, USA.
21. Laskin R. SIM dynamics and control requirement flowdown process. *Presentation at the SIM Project Preliminary Instrument System Requirements Review*, *JPL*, March, 1998; 17–18.
22. Bélanger P. *Control Engineering*: *A Modern Approach*. Saunders College Publishing: Fort Worth, 1995.
23. Elias L. *A Structurally Coupled Disturbance Analysis Method Using Dynamic Mass Measurement Techniques*, *with Application to Spacecraft—Reaction Wheel Systems*. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, March, 2001.
24. Zhou K, Doyle JC, Glover K. *Robust and Optimal Control*. Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1996.
25. Wirsching PH, Paez TL, Ortiz H. *Random Vibrations*: *Theory and Practice*. Wiley: New York, 1995.
26. Gelb A. *Applied Optimal Estimation*. The MIT Press: Cambridge, MA, 1974.
27. Chen TW, Francis B. *Optimal Sampled-Data Control Systems*. Communications and Control Engineering Series. Springer: London, 1995.
28. Yamamoto Y. A function space approach to sampled data control systems and tracking problems. *IEEE Transactions on Automatic Control* 1994; **39**(4):703–713.
29. Golub G, Van Loan C. *Matrix Computations* (3rd edn). The Johns Hopkins University Press: Baltimore, MD, 1996.