MULTIPLE FRAME RATE INTEGRATION

88-4579-CP

A. Haraldsdottir
Applied Dynamics International, Ann Arbor, Michigan
and
R. M. Howe*
The University of Michigan, Ann Arbor, Michigan

Abstract

Dynamic systems can often be separated into fast and slow subsystems. The speed and accuracy of a simulation of such systems can frequently be improved by using a frame rate for numerical integration of the fast system which is an integer multiple of the frame rate used for the slow system. The technique of multiple frame-rate integration can be especially important in real-time simulation. In this paper the multiple frame-rate method is introduced, including techniques for converting slow data sequence outputs from slow subsystems to fast data sequence inputs for fast systems. The suitability of various integration algorithms for multiple framing is discussed. The implementation of multiple frame-rate integration using the simulation language ADSIM for the AD 100 computer is described, including software which allows, without program recompiling, choice of multiple-frame ratios and choice of different interpolation or extrapolation algorithms for slow-to-fast system interfacing. The paper concludes with an example of multiple framing applied to the simulation of a combined air frame and flight control system in order to improve both the accuracy and stability of the simulation.

1.Introduction

Dynamic systems can often be separated into fast and slow subsystems. One example is a combined air frame and flight control system, where the rigid airframe represents a slow subsystem, and both elastic structural modes and the flight-control system, including control-surface actuators, represent fast subsystems. Another example is a helicopter when modeled by the blade element method, where the rigid airframe again is the slow subsystem and the rotors are fast subsystems.

Multiple frame rate integration refers to the technique of making an integer multiple of integration passes through one or more fast subsystems for each pass through the slow subsystem. This reduces the integration step size for the fast subsystem. Since the dynamic errors in a digital simulation will be dominated by the integration truncation errors associated with the fast subsystem, the use of multiple framing can improve significantly the simulation accuracy for a given real-time processor. The accuracy improvement when using multiframing is much more substantial when the fast subsystem is considerably less complex and therefore requires much less processor time than the slow subsystem.

The overall concept of multiple frame rate integration is described in Section 2, along with the requirement to use extrapolation or interpolation to interface slow subsystems to fast subsystems. The section also introduces dynamic error measures, following which the compatibility of specific integration methods with multiple framing is discussed. Section 3 presents various interpolation and extrapolation algorithms for slow to fast data sequence conversion, as well as the dynamic errors associated with this conversion process.

Often it may not be clear exactly how a dynamic system should be partitioned into fast and slow subsystems in order to make most effective use of multiple framing. It may also be difficult to predetermine the optimal frame rate multiple for dynamic accuracy improvement. Analytic methods based on both time and frequency domain considerations, as introduced in Section 2, help in making these choices. However, in Section 4 an interactive software system is described which permits the user to experiment with different problem partitioning, frame rates, and interface extrapolation and interpolation methods. In Section 5 a combined air frame and flight-control system is used to illustrate the multiple framing analysis and synthesis techniques described in the earlier sections. Section 6 contains the concluding remarks.

2. Description of Multiple Frame Rate Integration

The separation of a dynamic system into slow and fast subsystems is illustrated in Figure 1. The slow system utilizes an integration step size denoted by T, whereas the fast system employs a step size denoted by h, where $h = T/N$ and N is an integer. Hereafter we will refer to N as the frame ratio. In Figure 1 the output data sequence $\{r_n\}$ with sample period T from the slow subsystem is converted to a fast data sequence $\{f_k\}$ with sample period h by means of an interpolator (or extrapolator). This is necessary to provide the fast subsystem with inputs having a sample period h equal to the fast subsystem integration step size. Examples of the generation of a fast sequence from a slow sequence are shown in Figure 2 for a frame ratio $N = 4$. In Figure 2a first-order interpolation is illustrated; in Figure 2b first-order extrapolation is used. Clearly the interpolation gives a more accurate result than extrapolation. In Section 3 we will see how we can quantify the dynamic accuracy of these and other interpolation and extrapolation algorithms in terms of equivalent gain and phase shift for sinusoidal data sequences.

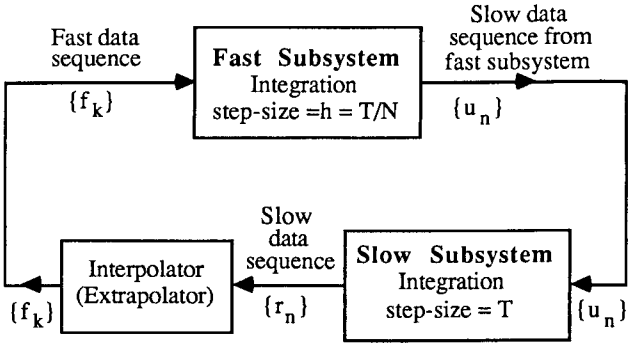

Figure 1. Multiple framing applied to a dynamic system.

*Professor, Department of Aerospace Engineering
Member AIAA

The output data sequence with sample period h generated by the fast subsystem in Figure 1 will need to be converted to a slow data sequence $\{u_n\}$ with sample period T in order to provide inputs for the slow subsystem integration algorithm. This conversion is easily accomplished by utilizing every Nth member of the fast data sequence output, although some multiple-pass integration methods may also require intermediate data points over the sample-period T.

From Figure 2 it is evident that interpolation of the data sequence $\{r_n\}$ over the nth sample period requires both r_n and r_{n+1}. Clearly r_{n+1} will only be available if the nth integration step in the slow subsystem has already been executed. On the other hand, extrapolation of the data sequence $\{r_n\}$ over the nth sample period requires r_n and r_{n-1}, both of which are available without prior execution of the nth integration frame in the slow subsystem. In this case, therefore, there is the option of performing the N fast subsystem integrations first with step size h, followed by the slow subsystem integration with step size T.

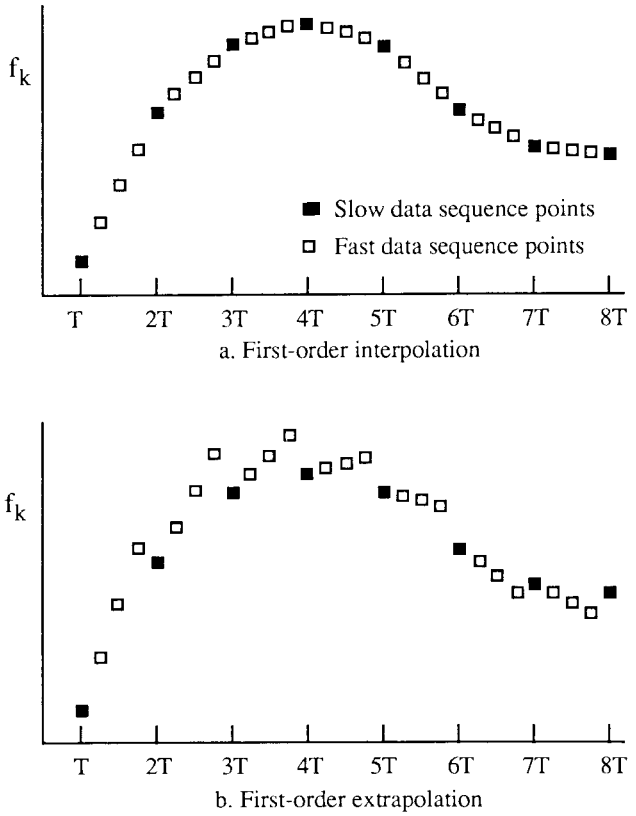

Figure 2. Generation of a fast data sequence from a slow data sequence. $N = 4$.

Before discussing the suitability of various numerical integration algorithms for multiple framing, we review briefly the formulas and attributes of several of the candidate integration methods for real-time simulation. We will also describe dynamic error measures and stability considerations which will be important in understanding the benefits to be derived from multiple framing.

One of the most commonly used algorithms for real-time integration is the second-order Adams-Bashforth predictor method. The AB-2 formula for integrating the state equation $dx/dt = f[x,u(t)]$ is given by

$$x_{n+1} = x_n + h\left(\tfrac{3}{2}f_n - \tfrac{1}{2}f_{n-1}\right) \tag{1}$$

where

$$f_n = f(x_n, u_n) \tag{2}$$

Here $x_n = x(nh)$ and $u_n = u(nh)$, where h is the integration step size, n is an integer, x is the state variable, and $u(t)$ is the input, which is considered to be an explicit function of time t. The AB-2 formula is derived from the area under a linear extrapolation of the derivative f from $t = nh$ to $t = (n+1)h$ based on f_n and f_{n-1}. It is a second-order integration method, with dynamic errors proportional to h^2. When AB-2 integration is used to solve linear differential equations, it can be shown from z transform theory that the fractional error, e_λ, in any characteristic root λ is given approximately by the formula[1]

$$e_\lambda = \frac{\lambda^* - \lambda}{\lambda} \cong -\frac{5}{12}(\lambda h)^2, \quad |\lambda h| << 1 \tag{3}$$

Here λ^* is the equivalent characteristic root of the digital system. To estimate the dynamic errors in using AB-2 integration for nonlinear differential equations, we can linearize the equation with respect to any reference or steady-state solution and then apply Eq. (3). For a given step size h the largest error will result from the characteristic root λ of largest magnitude. Thus Eq. (3) can be used to estimate the characteristic root errors in any simulation with known roots or eigenvalues λ. Or, if we have a given accuracy requirement on the roots, Eq. (3) permits us to establish the maximum allowable step size h when using AB-2 integration for that simulation.

Since Eq. (3) applies equally well for complex roots , it can also be used in this case to derive the following approximate formulas for the fractional error in root frequency, e_ω, and the damping ratio error, e_ζ[1]:

$$e_\omega = \frac{\omega^* - \omega}{\omega} \cong \frac{5}{12}(1 - 4\zeta^2)(\omega_n h)^2, \quad \omega_n h << 1 \tag{4}$$

$$e_\zeta = \zeta^* - \zeta \cong \frac{5}{6}(\zeta - \zeta^3)(\omega_n h)^2, \quad \omega_n h << 1 \tag{5}$$

Here ω, ζ and ω_n represent the frequency, damping ratio and undamped natural frequency, respectively, associated with the complex root λ, while ω^* and ζ^* are the frequency and damping ratio, respectively, associated with the equivalent root λ^* of the digital system. Again, for any complex root pair Eqs. (4) and (5) can be used to estimate the frequency and damping ratio errors. Alternatively, for required accuracy in root frequency and damping ratio, Eqs. (4) and (5) can be used to establish the maximum allowable step size h when using AB-2 integration for the simulation. Eqs. (3), (4) and (5) will be useful in establishing the optimal frame ratio N when using multiple framing.

AB-3 and AB-4 predictor algorithms, based on second and third-order extrapolation, respectively, of the state-variable derivatives, can also be used as real-time integration methods. They produce dynamic errors proportional to h^3 and h^4, and formulas equivalent to Eqs. (3), (4) and (5) for characteristic root errors can be derived[1].

In addition to characteristic root errors it is important to consider the stability of integration algorithms, especially when using predictor methods. This is because the extrapolations based on past state-variable derivatives introduce extraneous states and hence extraneous roots into the simulation. If the step size h becomes too large, these roots can cause instability, even though the principal roots are accurately represented. In Figure 3 the stability boundaries for the AB predictor methods are shown in the λh plane. For any combination of step size h and eigenvalue λ outside these boundaries the simulation will be unstable. This can also become a very important reason for considering the use of multiple framing, as we shall see later in the example in Section 5.

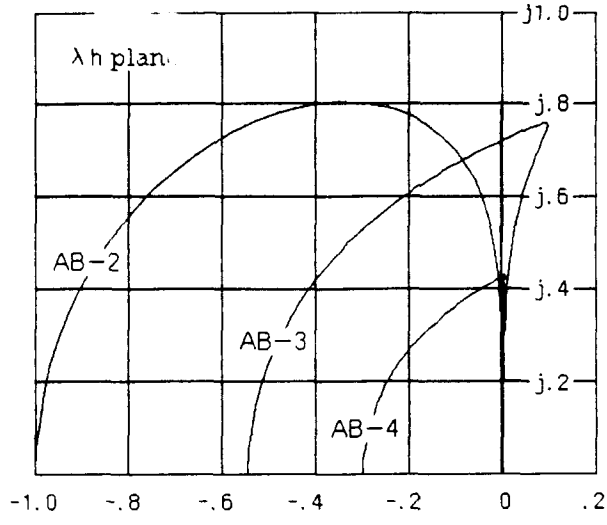

Figure 3. Stability boundaries for AB-2 integration.

In addition to the single-pass predictor-corrector algorithms considered above, real-time simulations can employ mutiple-pass integration algorithms such as Runge-Kutta methods. An example is the following RK-2 algorithm:

$$\hat{x}_{n+1/2} = x_n + \frac{h}{2} f(x_n, u_n) \tag{6}$$

$$x_{n+1} = x_n + hf(\hat{x}_{n+1/2}, u_{n+1/2}) \tag{7}$$

Euler (rectangular) integration is used for the first pass in Eq. (6) with a step size of $h/2$ to compute an estimate, $\hat{x}_{n+1/2}$, for the state halfway through the integration step. This estimate is then used in the second pass, along with the input $u_{n+1/2}$, to compute the derivative halfway through the step. In Eq. (7) this derivative is used to compute x_{n+1}. Both the AB-2 algorithm in Eqs. (1) and (2), and the RK-2 algorithm in Eqs. (6) and (7), are compatible with real-time inputs, since in both cases the input u is not required for algorithm execution prior to its availability in real time. For this reason the above version of Runge-Kutta integration is often designated by RTRK-2. The more commonly used version of RK-2, frequently called Heun's method, employs Euler integration with a step size of h for the first step. The resulting estimate, $\hat{x}_{n+1}$, is then employed in the second pass, along with u_{n+1}, to compute x_{n+1} using trapezoidal integration. Since u_{n+1} is not yet available in real time at the start of the second pass, standard RK-2 is not compatible with real time inputs.

The RTRK-2 formula for e_λ, the fractional eror in characteristic root, is identical with that in Eq.(3) for AB-2 except that the coefficient 5/12 is replaced by the coefficient 1/6. For complex roots the RTRK-2 formulas for e_ω and e_ζ, the damping ratio error, are identical with those in Eqs. (4) and (5) for AB-2 except that the coefficients 5/12 and 5/6 are replaced by 1/6 and 1/3, respectively[1]. Thus the characteristic root errors associated with RTRK-2 integration for a given step size h are 40 percent of the AB-2 root errors. However, RTRK-2 is a two-pass method and will therefore take roughly twice as long to execute per integration step as AB-2. Since the dynamic errors vary as h^2, doubling the step size will quadrupal the errors. This more than offsets the error coefficient advantage of RTRK-2 over AB-2.

Another two-pass method is AM-2, the Adams-Moulton predictor-corrector algorithm. Here the first pass uses AB-2 integration to compute the predicted state, $\hat{x}_{n+1}$, which is employed along with u_{n+1} to calculate the derivative estimate $\hat{f}_{n+1}$. The second pass then uses trapezoidal integration to compute the final x_{n+1}. For AM-2 integration the approximate formulas for the charcateristic root errors are exactly -1/5th those shown in Eqs. (4), (5) and (6) for AB-2 integration. AM-2 is not compatible with real-time inputs because, as in standard RK-2, it requires u_{n+1} at the start of the second pass. However, a modified AM-2 method which computes $\hat{x}_{n+1/2}$ in the first pass can be used with real-time inputs[2].

There are two-pass predictor-corrector methods of higher order, e.g., AM-3 and AM-4, as well as three and four-pass Runge-Kutta methods (RK-3 and RK-4), any of which can be useful method for simulating dynamic systems. However, as noted below, the multiple-pass methods can complicate considerably the effective use of multiple framing.

From discussion thus far it is apparent that single-pass AB predictor algorithms are completely compatible with the multiple framing concepts described at the beginning of this section and illustrated in Figures 1 and 2. When we consider multiple-pass algorithms, however, the compatiblity is not so clear. For example, in the two-pass RTRK-2 method the first pass in Eq. (6) consists of a half-step Euler integration. When this is executed in the slow system, the second pass for the slow subsystem will require an input $u_{n+1/2}$ from the fast subsystem, as evident from Figure 1 and Eq. (7). How is this obtained with multiple framing from the fast subsystem? Does the fast subsystem execute N/2 two-pass RTRK-2 steps? If it does, the necessary fast subsystem inputs, if derived by interpolation from the slow subsystem, will need to be based on the rather inaccurate first Euler estimate $\hat{r}_{n+1/2}$. The other choice is to use extrapolation based on r_n and r_{n-1}, but this also may have accuracy problems. A question also arises regarding the second $N/2$ steps for the fast subsystem. Are these initiated from halfway through the slow frame, or from the beginning of the slow frame? Actually, it is probably more appropriate to use a single-pass method such as AB-2 for the fast subsystem, even though RTRK-2 is used for the slow subsystem. However, many choices and questions still arise. When a four-pass method such as RK-4 is used, the choices for the fast subsystem integration methods become even more numerous, although high order methods have been successfully employed with multiple framing[3]. Nevertheless, it is the opinion of the authors that single- pass methods are in general the algorithms best suited to multiple framing and lead to straightforward, easily understood mechanizations.

3. Extrapolation and Interpolation

In this section we consider interpolation and extrapolation algorithms as utililized in multiple framing to generate a fast data sequence from a slow data sequence. We first consider extraploation of a slow data sequence $\{r_n\}$ based on r_n and r_{n-1}. Let the extrapolation interval be denoted by the dimensionless parameter a. The corresponding extrapolation time interval is aT, where T is the sample period of the slow data sequence. The for r_{n+a}, the representation of $r(nT+aT)$ based on linear extrapolation, is given by

$$r_{n+a} = r_n + a(r_n - r_{n-1}) \tag{8}$$

In the example of linear extrapolation shown in Figure 2b, $N = 4$ and a = 1/4, 1/2 and 3/4 to generate the intermediate fast data sequence points from the current and past slow data-sequence points. To analyze the dynamic errors associated with Eq. (8) we consider the extrapolator transfer function for sinusoidal input data sequences and compare it with the ideal extrapolator transfer function. This is easily accomplished by using z transforms[4]. The z transform of Eq. (8) is

$$R_a^*(z) = [1 + a(1 - z^{-1})]R^*(z) \tag{9}$$

from which the extrapolator z transform, $H_e^*(z)$, is given by

$$H_e^*(z) = \frac{R_a^*(z)}{R^*(z)} = 1 + a(1 + z^{-1}) \tag{10}$$

The extrapolator transfer function for sinusoidal input data sequences is given by $H_e^*(e^{j\omega T})$. After dividing this by the ideal extrapolator transfer function, $H(j\omega) = e^{j\omega aT}$, we obtain the following formula for the fractional error in the extrapolator transfer function:

$$\frac{H_e^*(e^{j\omega T})}{H_e(j\omega)} - 1 = e^{-j\omega aT}[1 + a(1 - e^{-j\omega T})] - 1 \tag{11}$$

If the step size T is sufficiently small, the fractional transfer function error represented by the right side of Eq. (11) will be a complex number small in magnitude compared with unity. In this case it can be shown that the real part of the complex number is approximately equal to the fractional error in transfer function gain, and the imaginary part is approximately equal to the transfer function phase error[5]. When the exponential functions are expanded in power series with higher order terms neglected, the following approximate formulas are obtained for the first-order extrapolator transfer function gain and phase errors:

$$\text{Fractional gain error} = e_M \cong \frac{a(1+a)}{2}(\omega T)^2 \quad , \quad \omega T << 1 \tag{12}$$

$$\text{Phase error} = e_A \cong 0 \cdot (\omega T)^2 \quad , \quad \omega T << 1 \tag{13}$$

Although Eqs. (12) and (13) are approximate, based on the dimensionless frequency ωT being small compared with unity, the formulas are surprisingly accurate up to $\omega T = 0.5$ for extrapolation intervals a over the range $0 < a < 1$. For the first order extrapolator it is clear that gain errors predominate over phase errors and are proportional to T^2.

Next we consider first-order interpolation based on r_n and r_{n+1}. Here the formula for r_{n+a} is given by

$$r_{n+a} = r_n + a(r_{n+1} - r_n) \tag{14}$$

Following the same procedure used for first-order extrapolation, we obtain the following formulas for the interpolator transfer function gain error:

$$\text{Fractional gain error} = e_M \cong \frac{a(a-1)}{2}(\omega T)^2 \quad , \quad \omega T << 1 \tag{15}$$

As in the case of extrapolation, the interpolation transfer function phase errors are zero to order T^2, i.e., the gain errors predominate. Over the range $0 < a < 1$, the interpolator gain error of Eq. (15) is significantly less than the extrapolator gain error of Eq. (12). This is also apparent in comparing Figures 2a and 2b.

Second-order interpolation and extrapolation formulas can also be derived and analyzed in terms of transfer function errors using the above methodology[5]. When this is done, it is found that the approximate gain errors are zero to order T^3; the phase errors are proportional to $(\omega T)^3$ and predominate.

The overall dynamic errors generated by the fast data sequence using interpolation or extrapolation can be analyzed in the frequency domain by considering a slow data sequence $\{r_n\}$ that is sinusoidal with frequency ω. This analysis shows that the fast data sequence consists of a component at the input frequency ω, as well as components at $2(N-1)$ additional frequencies. If ω_0 is the sample frequency for the slow data sequence ($\omega_0 = 2\pi/T$), these additional frequencies will occur at $\omega_0 \pm \omega$, $2\omega_0 \pm \omega$, ... , $(N-1)\omega_0 \pm \omega$. Fortunately the influence of the additional freqencies will in general be small. This is because their amplitude relative to the input sinusoid will be of the same order as the fractional gain or phase error of the interpolator (or extrapolator) transfer function[5]. Thus the principal component in the fast data sequence will be at the input frequency ω. Compared with the ideal interpolator (or extrapolator) output, this component will represent a gain or phase error which is the simple average of the N gain or phase errors associated with the transfer functions for the N interpolation (or extrapolation) intervals a.

In Table 1 the gain or phase error coefficients for slow-to-fast data sequence conversion when using first and second-order interpolation, and zero, first and second-order extrapolation are tabulated for frame ratios N = 2,3,4,5, and ∞. Note that zero-order extrapolation is equivalent to no extrapolation, i.e., the fast data sequence consists of the slow data sequence value r_n repeated N times, then r_{n+1} repeated N times, etc. In this case the phase error predominates and is proportional to ωT. For large N the phase error for zero-order extrapolation approaches $-\omega T/2$. This is equivalent to inserting a time delay between the slow and fast subsystems equal to one half of the slow subsystem integration step size T. It is apparent that failure to use extrapolation or interpolation in multiple framing has the potential of introducing substantial dynamic errors.

4. Implementation in the Simulation Language ADSIM

The simulation language ADSIM is a high level software system designed for the AD 100 simulation computer. The form of the ADSIM language is very similar to other continuous system simulation languages such as ACSL, CSSL, and CSMP.

Table 1. Multiple Frame Interpolator and Extrapolator Gain and Phase Errors

Note: N = frame ratio. Gain and phase errors based on $\omega T \ll 1$

Phase error coefficient, $e_A/(\omega T)$

Inputs	$N=2$	$N=3$	$N=4$	$N=5$	$N=\infty$
r_n (zero-order)	-.2500	-.3333	-.3750	-.4000	-.5000

Gain error coefficient, $e_M/(\omega T)^2$

	$N=2$	$N=3$	$N=4$	$N=5$	$N=\infty$
r_n, r_{n-1}	.1875	.2593	.2969	.3200	.4167
r_{n+1}, r_n	-.0625	-.0741	-.0781	-.0800	-.0833

Phase error coefficient, $e_A/(\omega T)^3$

	$N=2$	$N=3$	$N=4$	$N=5$	$N=\infty$
r_n, r_{n-1}, r_{n-2}	.1563	.2222	.2578	.2800	.3750
r_{n+1}, r_n, r_{n-1}	-.0313	-.0370	-.0391	-.0400	-.0417
$r_{n+1}, r_n, \dot{r}_n$	-.0104	-.0123	-.0130	-.0133	-.0139

Since ADSIM and the AD 100 have been designed to be especially effective for high-speed real-time simulation, integration with ADSIM is performed using fixed-step algorithms. The standard integration methods provided in the language are Euler, Adams-Bashforth 2,3 and 4, two-pass Adams-Moulton 2,3 and 4, Runge-Kutta 2 and 4, and real-time Runge-Kutte 2 and 3, as introduced in Eqs. (6) and (7). The default integration method used in ADSIM is AB-2. At run time the user can, without recompiling, interactively select any of the other integration methods for any or all of the state variables. He/she can also program his own integration algorithms as difference equations. In ADSIM the user writes differential equations in first-order state-variable form. The equations are solved in a segment of code termed a DYNAMIC BLOCK. The user code in the DYNAMIC BLOCK is invoked by a routine called SIMEXEC, which executes on the AD 100 computer and performs execution control and integration.

To implement multiple frame rate integration on the AD 100 and make it as easy to use as possible, the routine SIMEXEC has been modified to handle the additional execution control needed. Since initialization and integration in ADSIM is performed on an entire DYNAMIC BLOCK, the modified routine SIMEXEC expects the differential equations to be divided into two blocks, one corresponding to the slow subsystem and the other to the fast subsystem. During each simulation frame the right hand sides of all equations in the slow DYNAMIC BLOCK are evaluated and integrated once, while equations in the fast DYNAMIC BLOCK are evaluated and integrated N times, where N is the frame ratio. The user can interactively control N at run time. Figure 4 illustrates the execution flow of the modified SIMEXEC routine. The code segments "sync1" through "sync7", "initial" and "terminal" represent optional code to allow the user to perform special calculations before and after the simulation is performed. The routine $DER evaluates the state-variable derivatives in the dynamic equations, and $INT and $INC perform integration of continuous equations and updating of discrete-time (i.e., difference) equations.

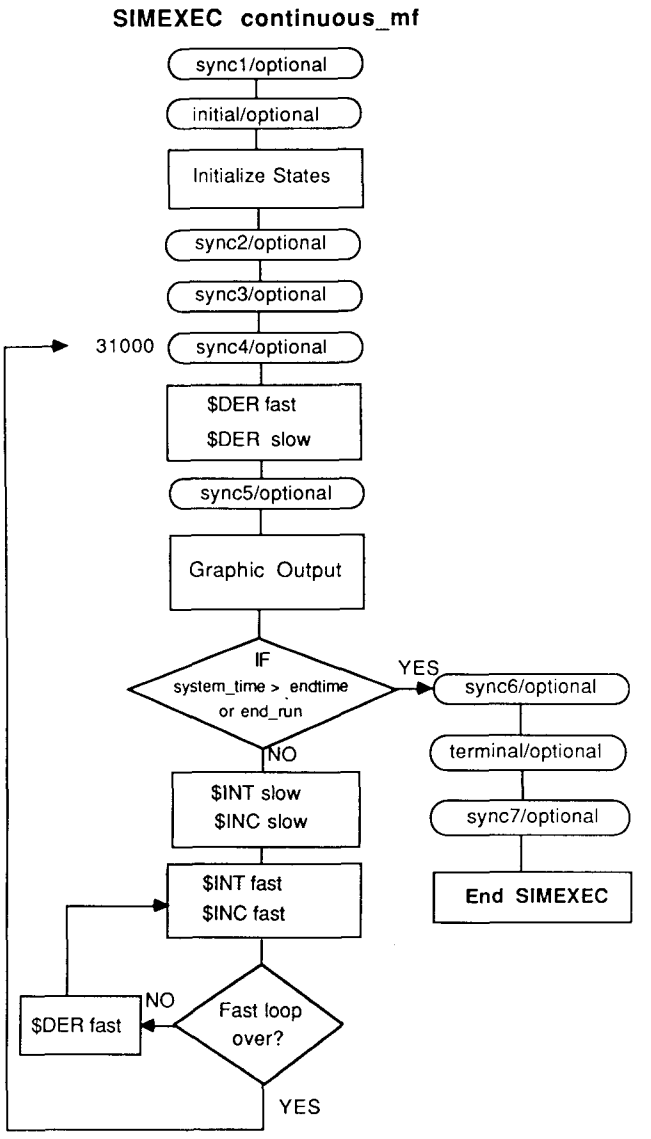

Figure 4. SIMEXEC for multiple frame rate integration.

In the dynamic block containing the fast subsystem the user needs to invoke a subprogram to perform interpolation on each of the slow variables used as inputs to the fast subsystem. Five subprograms for interpolation are supplied as standard functions with the ADSIM software package. Figure 5 illustrates the ADSIM code required to simulate a fourth-order system with multiframing.

To summarize, the user wishing to simulate a dynamic system using multiple frame rate integration needs only supply:

- The dynamic equations separated into slow and fast subsets.
- A subprogram invocation for each slow state variable to be interpolated.
- The number of fast integration steps for each slow step, i.e., the frame ratio N.

```
TITLE Two time scale model with multiframing
!
REGION initial
      frame_ratio_inv = 1./frame_ratio
END REGION
!
DYNAMIC fast
x3_inter = INTN1N(frame_ratio_inv,     %
                  slow_sample,x3,initialize)
e     =  r - kf*x3_inter
x1'   =  x2
x2'   =  -2.*z1*w1*x2 - w1*w1*x1 + k1*e
END DYNAMIC
!
DYNAMIC slow
x3'   =  x4
x4'   =  -2.*z2*w2*x4 - w2*w2*x3 + k2*x1
END DYNAMIC
!
DATA   z1          =   .3     %
       w1          =   35.    %
       k1          =   1000.  %
       z2          =   .9     %
       w2          =   .3     %
       k2          =   1.     %
       kf          =   1.     %
       r           =   1.
!
EXECUTE continuous_mf
```

Figure 5. ADSIM code to simulate a fourth-order system using multiframing.

The interactive real-time simulation features of ADSIM are preserved in the multiframing code. The variable "frameratio" can be changed interactively, and the actual computer execution time for one simulation frame is automatically measured and displayed after each change. The integration step size T divided by the frame execution time represents the execution speed with respect to real time (i.e., speedup over real time). This ratio is also calculated automatically after each change in multiframing parameters. When the ratio exceeds unity, the user knows the problem can be run in real time.

5. Multiframing Example

The use of multiple frame rate integration will be illustrated here with a reasonably large model that contains a small subsystem with fast dynamics. The model represents a typical business jet aircraft and includes a large data base associated with a number of multivariable aerodynamic functions, as well as a simplified flight control system. The aircraft is modeled as a rigid body with six degrees of freedom, three translational and three rotational. For the purpose of our multiframing example, only the aircraft pitch control loop will be considered. Figure 6 shows a block diagram of this control loop, where the inner feedback loop with the state variables δ_{es} and $\dot{\delta}_{es}$ is associated with the elevator control surface actuator.

Even though the overall airframe/flight-control system equations are nonlinear, it is useful to know the eigenvalues of the linearized equations in order to apply the stability and characteristic root error analysis discussed in Section 2. The eigenvalue computation is accomplished as follows: With the overall simulation frozen at any time, each state variable is perturbed by a small increment. From the resulting change in the state variable time derivatives, the partial of all state derivatives with respect to all state variables is computed numerically. From the Jacobian thus obtained the eigenvalues of the system are calculated. For a test flight condition of Mach 0.72 at 40,000 feet altitude the following eigenvalues were obtained:

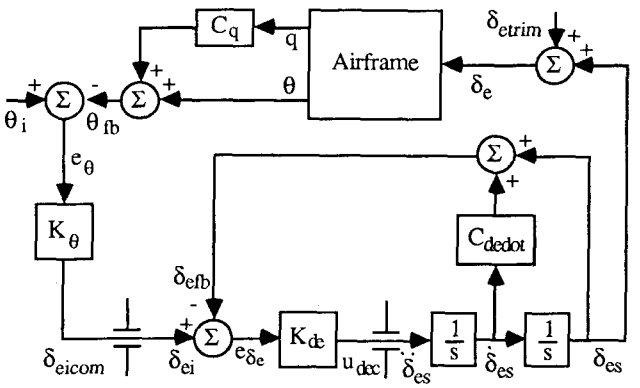

Figure 6. Pitch control loop.

$\lambda_{1,2} = -15.92 \pm j26.37$ $(\omega_n = 30.80, \zeta = 0.517)$
$\lambda_{3,4} = -1.274 \pm j4.674$ $(\omega_n = 4.844, \zeta = 0.263)$
$\lambda_5 = -1.000$
$\lambda_6 = -0.338$
$\lambda_7 = -0.016$

In the elevator actuator loop $C_{dedot} = 0.033$ and $K_{de} = 1000$. This makes the actuator natural frequency equal to 31.62 rad/s and the damping ratio 0.522. When the pitch-control loop is closed, this is the origin of $\lambda_{1,2}$ pair shown above. The other roots, which originate from the airframe poles and the quaternion stabilization loop, are substantially lower in magnitude. Clearly the pitch control loop is the fast subsystem which should benefit from multiple framing. Thus the partitioning of the overall model into fast and slow subsystems is straightforward in this example. In Figure 6 the airframe block represents the slow subsystem with the rest of the diagram representing the fast subsystem. The pitch loop equations are incorporated in the fast DYNAMIC BLOCK along with the subprogram invocations for the slow state variable interpolations, as shown in Figure 7. Here the state variables are *dedot* and *des*, and the fast subsystem inputs for interpolation are the pitch rate, *q*, and the four quaternions, *e1*, *e2*, *e3*, and *e4*. The quaternions are used to calculate the direction cosines *a13*, *a23*, and *a33*, from which the pitch angle *theta* (i.e., θ), is computed. If Euler angles rather than quaternions had been used in the simulation, then only *q* and *theta* would need to have been interpolated.

```
DYNAMIC fast
!
!       Elevator actuator:
!
!       Pitch control system:
!
deicom  =  Ktheta*(thetai-theta-Cq*q_int)
dei     =  Deilim*SAT(deicom*Ideilim)
e1_int  =  INTN1N(frame_ratio_inv,slow_sample,%
                  e1,initialize)
e2_int  =  INTN1N(frame_ratio_inv,slow_sample,%
                  e2,initialize)
e3_int  =  INTN1N(frame_ratio_inv,slow_sample,%
                  e3,initialize)
e4_int  =  INTN1N(frame_ratio_inv,slow_sample,%
                  e4,initialize)
e1e1    =  e1_int*e1_int
e2e2    =  e2_int*e2_int
e3e3    =  e3_int*e3_int
e4e4    =  e4_int*e4_int
a13     =  2.*(e2_int*e4_int-e1_int*e3_int)
a23     =  2.*(e2_int*e3_int+e1_int*e4_int)
a33     =  e1e1+e2e2-e3e3-e4e4
theta   =  ATAN2(-a13,SQRT(a23*a23+a33*a33))
q_int   =  INTN1N(frame_ratio_inv,slow_sample,%
                  q,initialize)
udec    =  Kde*(dei-des-Cdedot*dedot)
dedot'  =  Udelim*SAT(udec*Iudelim)
des'    =  dedot

END DYNAMIC
```

Figure 7. The fast DYNAMIC BLOCK in the multiframe simulation.

The benefit of multiple frame rate integration on both solution accuracy and numerical stability will be demonstrated here. Several interpolation algorithms were used for generating the fast data sequences, ranging from zero-order (no interpolation) to second order formulas. To study the accuracy improvement with multiframing, a reference solution was generated for pitch flight control response with an integration step size $T_1 = 0.000050$s. The pitch angle input θ_i was a step at $t = 0$ with a magnitude of 0.15 radian. AB-2 integration was used. Figure 8 shows the time history of the pitch angle response, θ, and the elevator actuator output, δ_{es}.

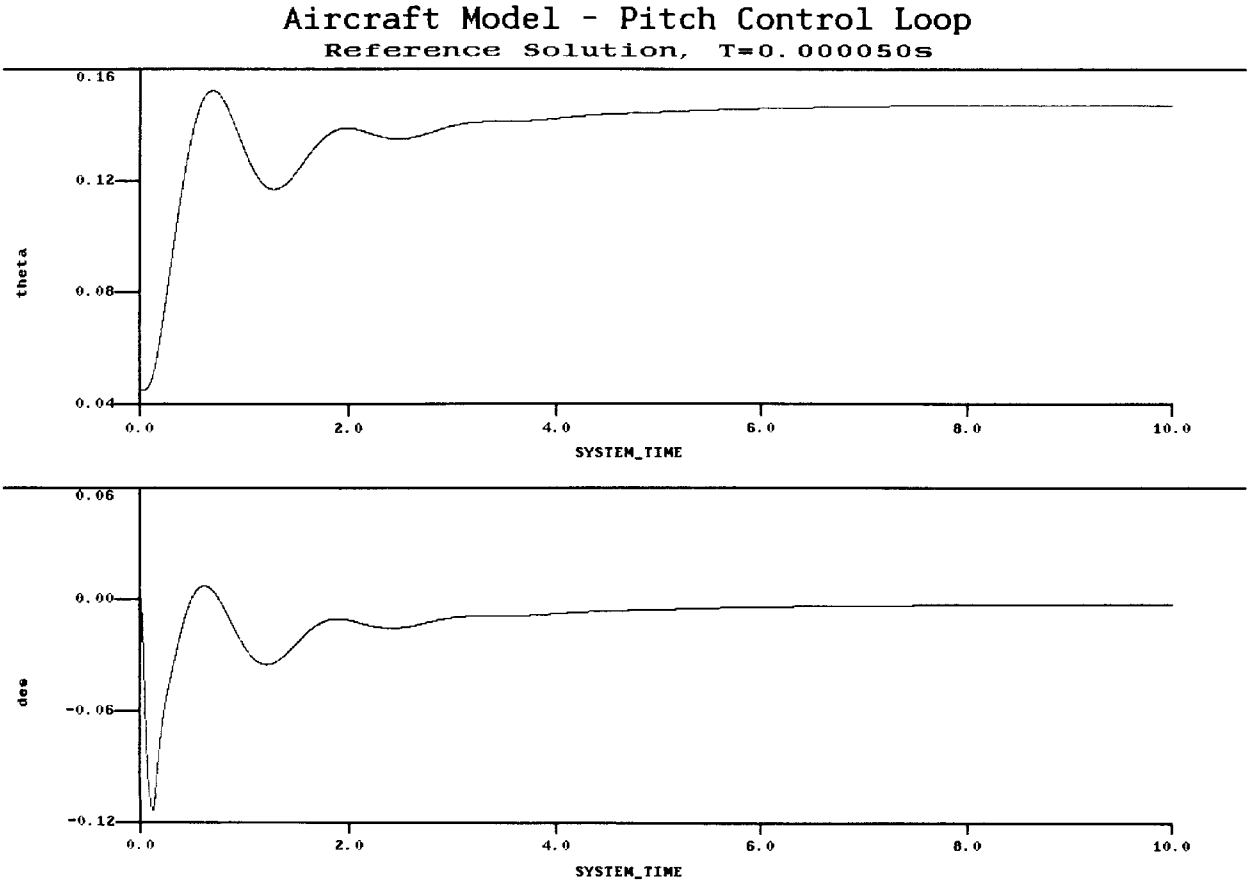

Figure 8. Reference step response solution for the pitch control system.

Time histories for a step size T_2 = 0.01s were then generated for frame ratios N ranging from 1 (no multiple framing) to 7. The error measure was computed as the mean absolute error in the elevator actuator output, δ_{es}. Table 2 shows the solution error for different frame ratios. It is clear from this data that multiple frame rate integration improves the solution accuracy substantially for frame ratios N up to 4. Beyond $N = 4$ there is little improvement.

Table 2. Solution Error for Different Frame Ratios with AB-2 Integration; Interpolation Based on r_{n+1} and r_n.

Frame Ratio	Mean absolute error in δ_{es}
1 (no framing)	0.0004725 rad
2	0.0001461
3	0.0000952
4	0.0000821
5	0.0000794
6	0.0000792
7	0.0000791

The results in Table 2 can be understood by applying Eqs. (4) and (5) for the AB-2 characteristic root errors to the system eigenvalues listed earlier in this section. When this is done for $\lambda_{1,2}$ = -15.92 ± j 26.37 with a step size h = 0.01, the predicted fractional error in frequency e_ω = -0.00273 and the damping ratio error e_ζ = 0.0300. For $\lambda_{3,4}$ = -1.274 ± j4.674 and h = 0.01, e_ω = 0.00071 and e_ζ = 0.00048. Thus when no multiframing is used, the errors for eigenvalues $\lambda_{1,2}$ associated with the pitch control loop are much larger than the errors for the eigenvalues $\lambda_{3,4}$ associated with the airframe.

On the other hand when $N = 5$, the pitch control loop has a step size given by 0.01/5 = 0.002. Applying Eqs. (4) and (5) to $\lambda_{1,2}$ with h = 0.002 gives e_ω = -0.00011 and e_ζ = 0.0012. The larger of these, e_ζ, is now roughly comparable with the e_ω (= 0.00071) and e_ζ (= 0.00048) as obtained above for the roots $\lambda_{3,4}$ associated with the airframe system, which still utilizes the step size 0.01. For N greater than 5 the airframe roots will actually predominate and nothing is gained by multiframing the pitch control loop to reduce its step size.

To examine the effect of multiple frame rate integration on the numerical stability, the aircraft simulation was executed using AB-2 integration with five different interpolation algorithms and various frame ratios. In ADSIM the variables "frametime, "steptime" and "speedup" are used to show the relationship between integration step size and real-time execution. "Frametime" is the actual time in seconds required for the AD 100 to execute a single pass through the dynamic equations, i.e., one pass through the slow subsystem and N passes through the fast subsystem. "Steptime" is the mathematical step size used for the integration step T in the slow subsystem. "Speedup" is therefore defined as

speedup = steptime/frametime

and represents the speedup factor over real time in the AD 100 solution for the stepsize T being used.

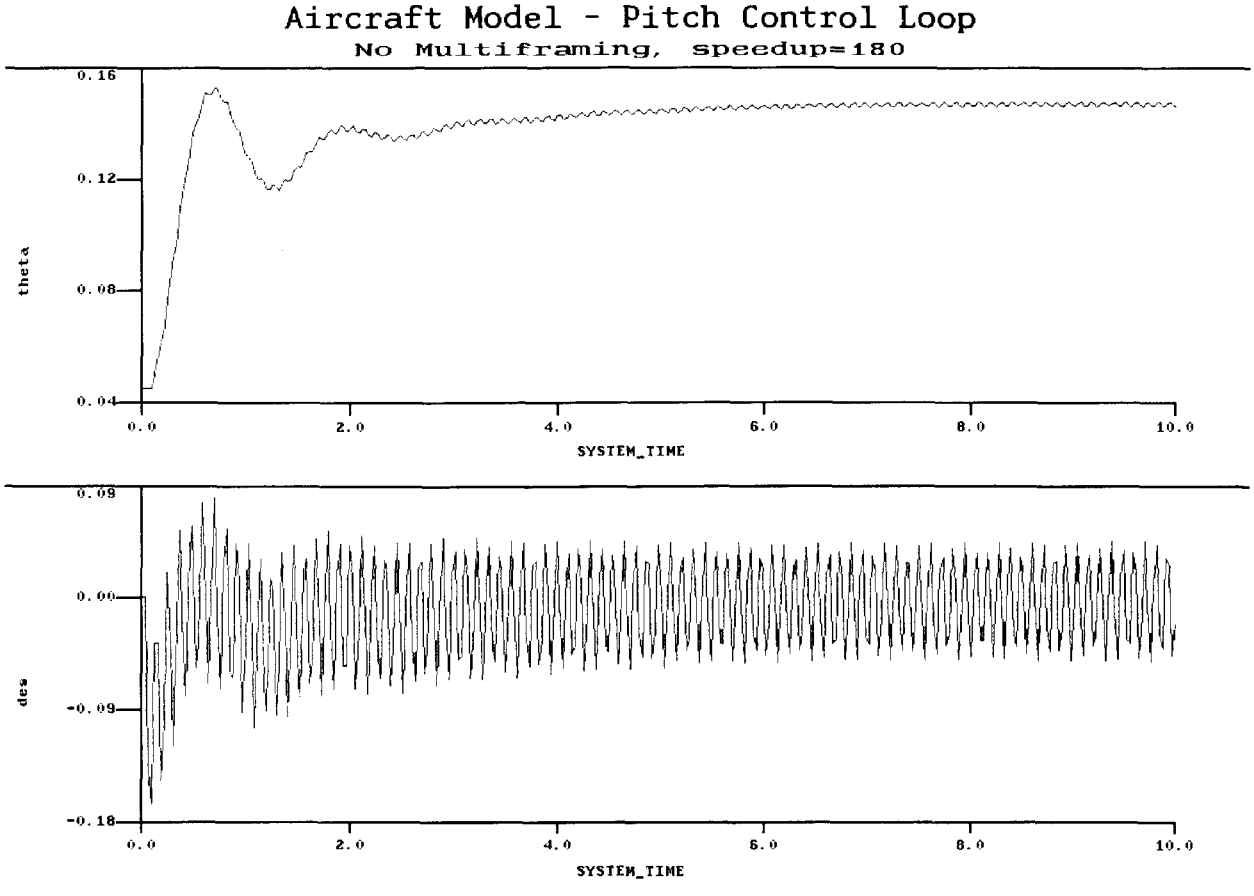

Figure 9. Marginally stable solution.

In the numerical stability experiment, marginal stability was determined by observing a graphical output of the pitch angle θ in response to the step input $\theta_i = 0.15$ radian. Figure 9 shows a typical time history for a marginally stable solution with the frame ratio $N = 1$, i.e., no multiple framing. Table 3 lists the integration step size T and corresponding speedup factor at which the solution becomes marginally stable for frame ratios ranging from 1 to 7 and for five different interpolation algorithms. Note that interpolation based on r_n is zero-order and hence represent no interpolation.

Comparison of the performance in Table 3 for various interpolation algorithms shows that first-order interpolation based on r_{n+1} and r_n gives the best performance. The improved dynamics of the higher-order interpolation algorithms is more than offset by their increased execution time. It is also clear that zero-order interpolation based on r_n, i.e., non interpolation, produces inferior results. This is because of the additional time delay, $T/2$, introduced between the slow and fast subsystems by zero-order interpolation, as noted in Section 3.

Table 3. Integration Step Size and Speedup Factor for Marginally Stable Solution.

Basis for interpolation	Integration step size T (speedup over real time)						
	Frame ratio $N = 1$	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$	$N = 7$
r_n	.02968 (187)	.05826 (306)	.08307 (374)	.1003 (395)	.1114 (390)	.1158 (365)	.1169 (335)
r_{n+1}, r_n	.02987 (183)	.05838 (287)	.08722 (361)	.1150 (411)	.1437 (452)	.1539 (432)	.1578 (400)
r_{n+1}, r_n, r_{n-1}	.02976 (169)	.05833 (259)	.08640 (315)	.1155 (357)	.1442 (387)	.1522 (361)	.1520 (323)
$r_{n+1}, r_n, \dot{r}_n$	.02978 (176)	.05814 (275)	.08673 (342)	.1154 (390)	.1437 (425)	.1521 (400)	.1521 (360)
$r_{n+1}, r_n, \dot{r}_n, r_{n-1}$	.02980 (168)	.05832 (256)	.08652 (311)	.1153 (351)	.1440 (380)	.1524 (335)	.1545 (322)

For $N = 1$ (no multiple framing) the table shows that the system becomes marginally stable for $T = h = 0.02968$. Here the instability is due to the eigenvalues $\lambda_{1,2} = -15.92 \pm j\ 26.37$, as listed earlier in this section and associated with the dynamics of the actuator. Thus $\lambda_1 h = -0.473 + j\ 0.783$, which indeed lies on the AB-2 stability boundary in Figure 3. This confirms that the instability with no multiframing originates in the fast subsystem. From z transform theory it turns out that the predicted frequency of instability in this case is equal to $0.270/T$ or 9.10 hertz, which is precisely the frequency of the undamped transient in Figure 9. On the other hand, for $N = 7$ and interpolation based on r_{n+1} and r_n, the system becomes marginally stable for $T = 0.1578$. In this case the step size of the fast subsystem is given by $h = T/7 = 0.0225$, which lies well within the AB-2 stability boundary for its roots $\lambda_{1,2}$. Thus the instability must result from the roots $\lambda_{3,4} = -1.274 \pm j\ 4.674$ associated with the slow subsystem. In this case $\lambda_3 T = -0.201 + j\ 0.738$, which indeed lies just within the AB-2 stability boundary in Figure 3 and confirms the expected instability based on the slow subsystem eigenvalues.

When multiframing is used for the fast subsystem, the "frametime" will actually increase. This is because the time required for the input interpolations and N integration steps of the fast subsystem is added to the execution time for the single integration step of the slow subsystem. Even though the overall frame time has increased, however, the integration step h for the fast subsystem will be reduced as the frame ratio N is increased, since $h = T/N$. This accounts for the improved stability evident in Table 3 for frame ratios up to $N = 5$, as reflected by the increased speedup over real time before the simulation becomes marginally stable. Beyond $N = 5$ the fast subsystem step size h has been reduced to the point where it is the slow system step size T that causes the instability. This is why the overall stability deteriorates for N larger than 5, since as N is further increased, the overall frame time and hence T for the slow subsystem actually increases.

In the example of mutiple frame rate integration described in this section, we have only considered interpolation for interfacing the fast subsystem to the slow subsystem. Alternatively, extrapolation could have been used. However, reference to Table 2 shows that the dynamic performance of extrapolation is considerably poorer. It should probably be considered only when it is desirable to avoid the necessity of integrating the slow subsystem first. There would, however, have been one advantage in using extrapolation for our example here. If extrapolation had been used, it would only have been necessary to interface the pitch angle θ and pitch rate q from the slow to the fast subsystem, rather than the four quaternions e_1, e_2, e_3 and e_4, as well as q. This would this have saved not only three interpolation calculations but also the additional calculation of direction cosines and θ each fast frame. When the interface between slow and fast subsystem consists entirely of state variables, this advantage for extrapolation is of course no longer present.

6. Conclusions

In this paper we have seen how the use of multiple frame rate integration in the simulation of a dynamic system can improve both the accuracy and stability of the simulation. In many cases it can make the difference between whether or not a simulation on a given computer can be run in real time with satisfactory accuracy. Analytic methods based on linearization of the dynamic system being simulated can be used to estimate the dynamic performance and stability for given integration methods and step sizes. This can in turn be utilized to make preliminary assessments of the optimal separation into slow and fast subsystems, as well as the optimal ratio N of the fast compared with the slow subsystem. However, it is also important to have an interactive software system which permits the user to experiment with the simulation itself in order to

determine optimal problem partitioning and frame ratios. The application of such a system, ADSIM for the AD 100 computer, to the simulation of a combined airframe, flight control system with the use of multiple framing has been described. An additional consideration of importance in mutiple framing is the type of interpolation or extrapolation algorithm used for the slow-to-fast system interfacing. Again, analytic techniqes have been described here for assessing the performance of various algorithms, although interactive experimentation with different interpolation or extrapolation methods at run time can also be extremely useful.

7. References

1. Howe, R.M., "Transfer Function and Characteristic Root Errors for Fixed-Step Integration Algotithms," *Trans. of the Society for Computer Simulation*, Vol.2, No. 4, Dec., 1985, pp 293-320.

2. Howe, R.M., "The Role of Modified Euler Integration in Real-time Simulation," *Proceedings of the Conference on Aerospace Simulation III*, San Diego, 1988; pp 277-285. The Society for Computer Simulation, P.O. Box 17900, San Diego, CA 92117.

3. Palusinski, O.A., "Simulation of Dynamic Systems Using Multirate Integration Techniques," *Trans of the Society for Computer Simulation*, Vol.2, No. 4, Dec., 1985, pp 293-320.

4. Gilbert, E.G., "Dynamic Error Analysis of Digital and Combined Digital-Analog Systems," *Simulation*, Vol. 6, No. 4, April, 1966, pp 241-257.

5. Howe, R.M., "Dynamics of Digital Extrapolation and Interpolation," *Trans. of the Society for Computer Simulation*, Vol.2, No. 1, Dec., 1985, pp 169-187.