

A Highly Efficient, General-Purpose Approach for Co-Simulation with ADAMS®

Andrew S. Elliott, Ph.D.
Mechanical Dynamics, Inc.
6530 East Virginia Street
Mesa, Arizona, USA 85215-0736
480.985.1557 aelli@adams.com

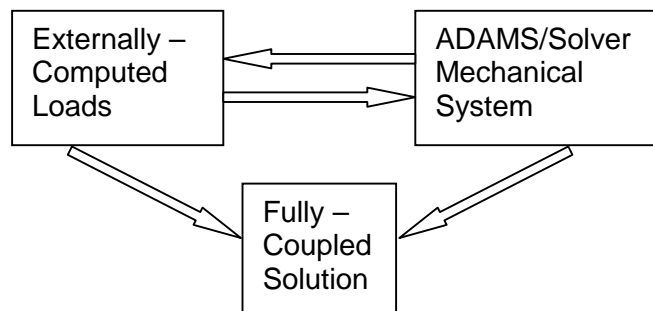
ABSTRACT

This paper presents a computationally efficient, general-purpose methodology for connecting any external computer code to ADAMS, in such a way that the two programs can communicate during system simulation and so that both programs can run at close to their best stand-alone speed. The method allows for the external code to run in discrete time if so designed, and it may have multiple and/or variable sampling rates. Two examples of problems that might take advantage of this methodology are aeroservoelastic response, where the aerodynamic loads are computed in a finite difference CFD code, and interactive simulation modification such as hardware-in-the-loop, where the user needs to change simulation inputs while the problem is running. The method uses a quadratic interpolation/extrapolation routine which is presented in FORTRAN, but which is easy to implement in any programming language.

INTRODUCTION

In advanced applications of ADAMS' mechanical system simulation technology, it often occurs that a user already has a sizeable investment in another computer code that solves part of the problem, for example to compute hydraulic, electromagnetic or aerodynamic loads. This may be a commercially available program or a specially developed, highly proprietary in-house product. This other code will generally have a solution methodology very different from ADAMS', and may be running on different hardware or at a different site. For these reasons, it is usually not possible to convert the other code to run as an ADAMS/Solver subroutine, nor to convert ADAMS/Solver to run as a subroutine of the other code.

In these cases, we need to be able to connect the other code to ADAMS/Solver in such a way that the two programs can communicate with each other during the system simulation to get a fully coupled response. Further, because this kind of problem tends to be large and complex, the connection needs to be made in an efficient manner which will allow both programs to run at the best possible speed, unless there are no constraints on time and hardware availability.



CO-SIMULATION CONSIDERATIONS

When setting up a co-simulation solution, it is important to consider these points:

1. ADAMS/Solver solves the system equations in continuous time and produces a continuous result. This is true even though we may choose to request output from Solver only in fixed time increments. Further, the solution methodology in ADAMS/Solver is arranged in such a way that internal simulation time may actually go backwards when the corrector is having trouble converging.
2. The other code you want to connect to Solver will often solve its part of the problem in discrete time and the result is not available between time steps. This can be true if the other code is using a discrete approximation to continuous time, e.g. a finite difference approach, or if the other code is actually modeling a discrete process, such as a digitally-controlled actuator.
3. The other code may be much slower, or much faster than ADAMS/Solver, depending on the relative complexities of the portions of the problems they are each solving, and depending on the computer hardware they are using. If possible, we would like to ensure that during the co-simulation each code can run at close to its best possible speed, and that the communications between the codes are not the limiting speed factor.

AN EXAMPLE PROBLEM – WHAT CAN GO WRONG

Using a very simple demonstration problem, we can show many of the things that can go wrong with a poorly arranged co-simulation. We will later use this same problem to show how we can greatly improve the results.

The model includes a pair of identical spring-mass-damper elements with the following characteristics:

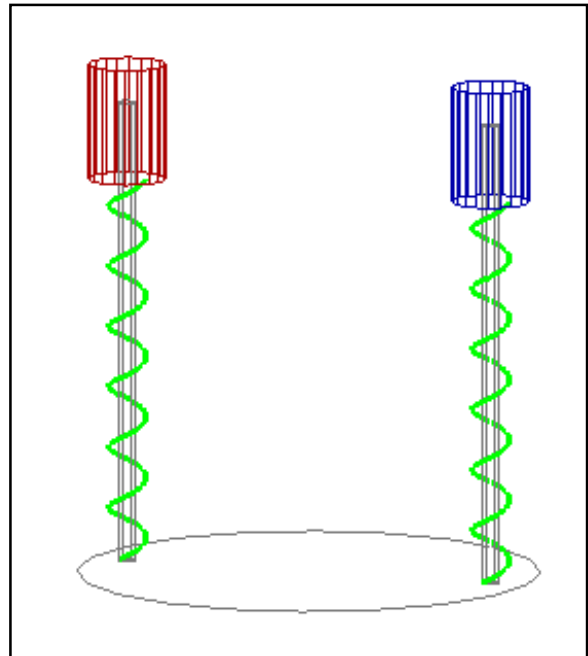
$$\begin{aligned}K &= 314.16 \text{ lbf/in} \quad (= 121391 \text{ lbm-in/sec}^2/\text{in}) \\C &= 0.0005 \text{ lbf-sec/in} \\M &= 0.3183 \text{ lbm}\end{aligned}$$

This is specifically set up to give a very stiff, nearly undamped system, whose natural frequency is close to 100 Hz (actually about 98.3 Hz).

One spring-mass-damper is forced by a regular ADAMS SFORCE element, using low frequency, purely sinusoidal forcing with a magnitude of 50 lbf and a period of 1 second.

The other spring-mass-damper will be forced by an external, discrete time code which approximates the same force. We can easily control the time step in this other code.

All simulations will run for 4 seconds.



There are a variety of things that can go wrong with a co-simulation problem. The most common problem is that there will be a communications bottleneck between the codes and the combined solution will just run very slowly. This kind of slowdown can also be caused by difficulties that one code has in “digesting” the data provided by the other code.

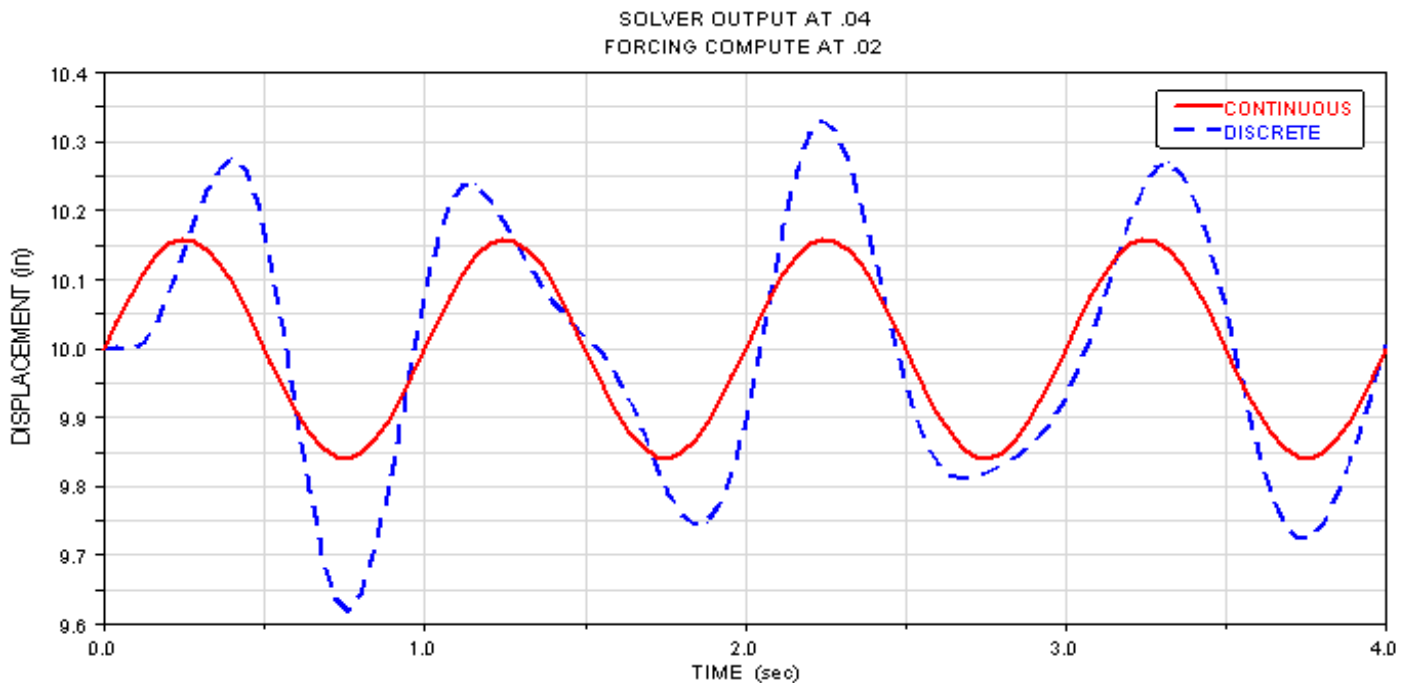
Certain kinds of problems, however, can lead to the co-simulation producing wrong answers. These problems include

- synchronization failure between the two codes
- aliasing due to inappropriate sampling interval
- numerical “pinging” in ADAMS caused by discrete inputs
- artificial instability caused by incompatible error control

Now let’s look at what happens to this simple system when we co-simulate it with various times steps for ADAMS/Solver and for the other code.

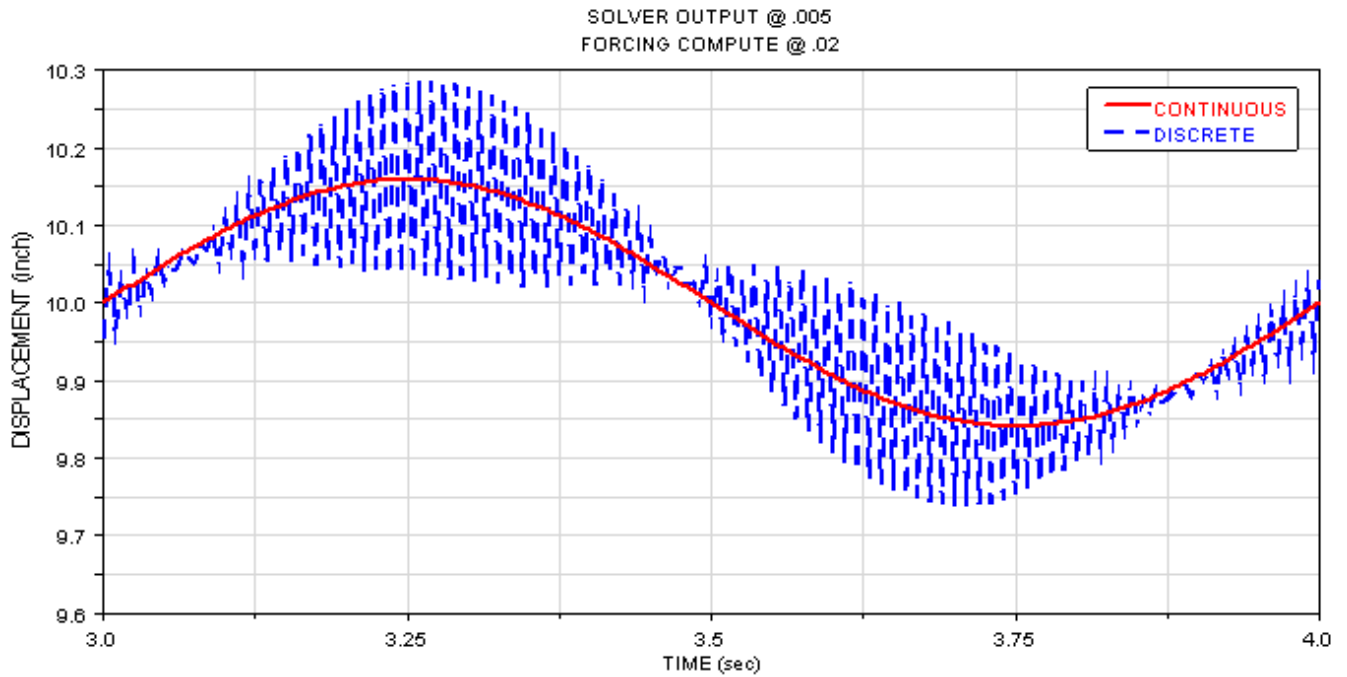
ADAMS/Solver – 25 output steps/sec (step size = .040)
External Force - 50 compute steps/sec (step size = .020)

Since the forcing frequency is only 1 Hz, we would expect these values to be more than sufficient to give very good results. The following plot compares the displacements of the mass with true continuous forcing to the that with co-simulated discrete forcing.



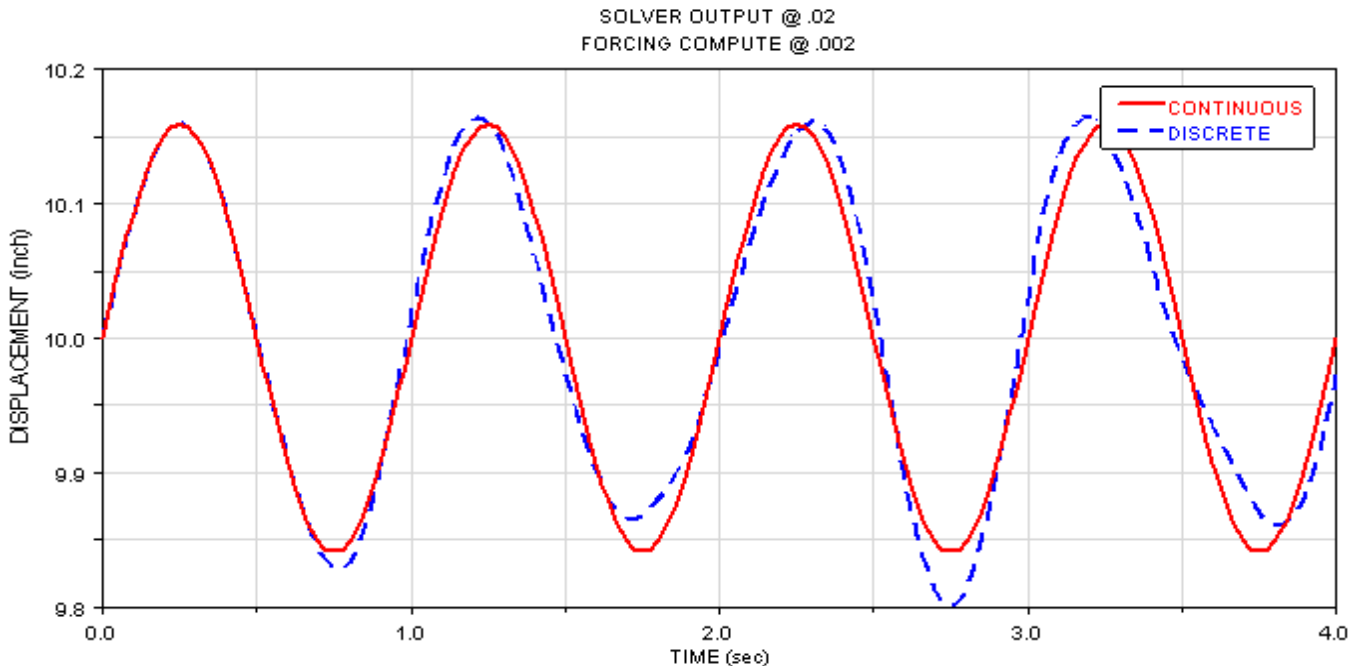
Decreasing the Solver step size to .02 or even .01 seconds has very little effect on the co-simulated response except for increasing the run time 10 or 20%. However, going to a .005 second or smaller step size for Solver allows the discrete forcing to numerically excite the 100 Hz mechanical system as shown in the following plot.

ADAMS/Solver – 200 output steps/sec (step size = .005)
External Force - 50 compute steps/sec (step size = .020)

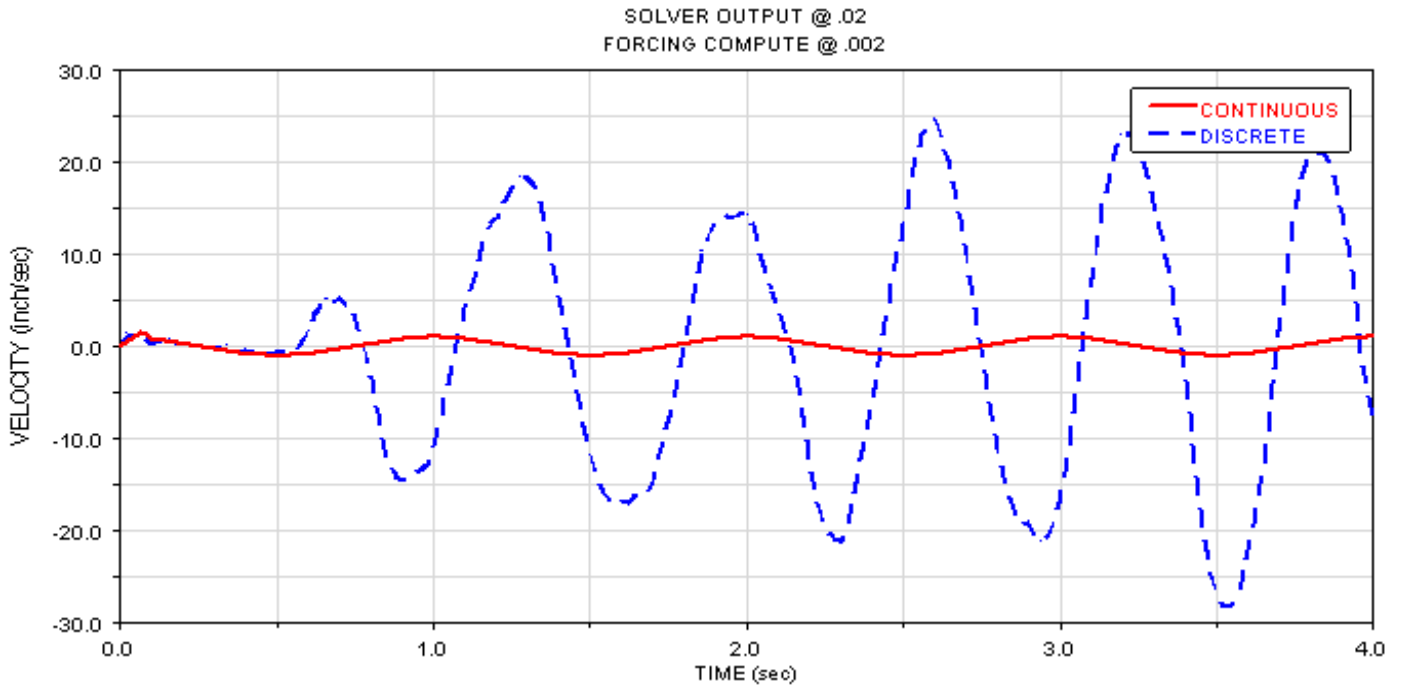


So this is apparently not a route to the desired solution. Instead, we can try to decrease the time step for the external forcing code to see if that improves the system response.

ADAMS/Solver – 50 output steps/sec (step size = .020)
External Force - 500 compute steps/sec (step size = .002)

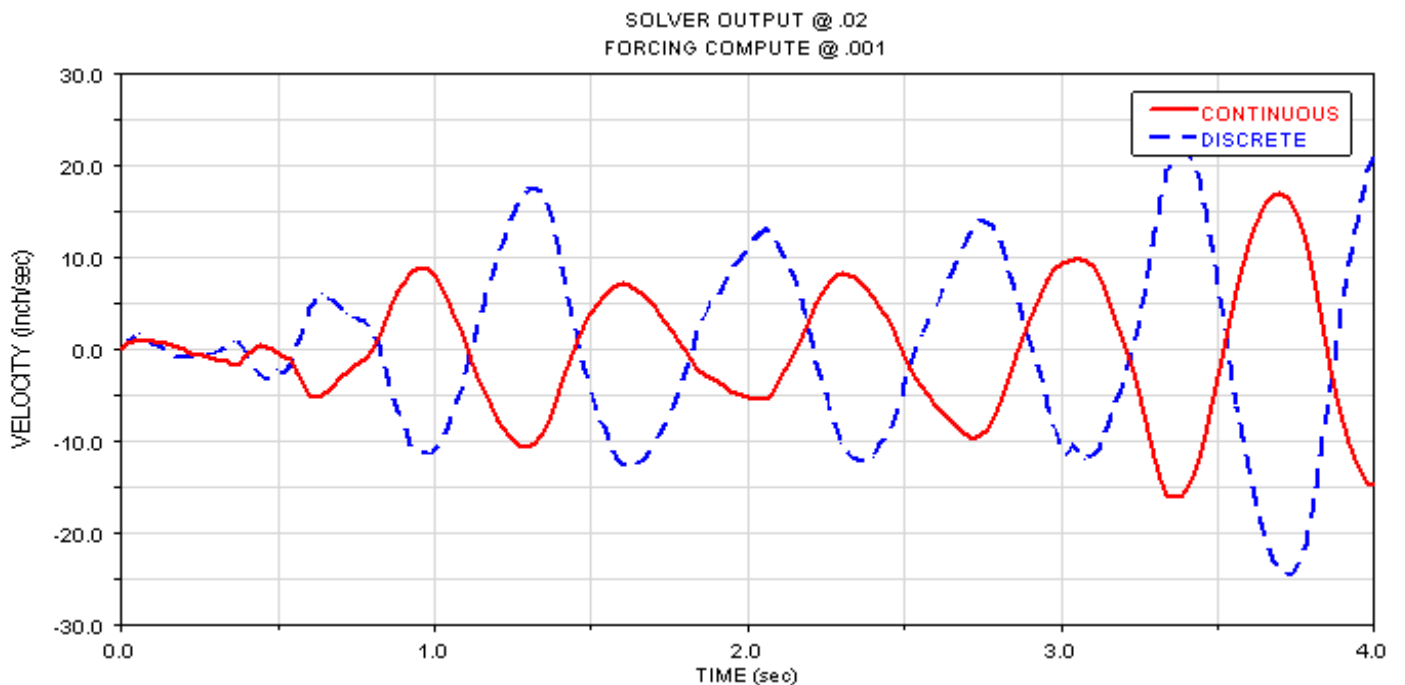


This would seem like a promising approach, but the response seems to be degrading as the simulation continues. If we look at the velocity traces, we see that a purely artificial instability has been introduced into the system at the unexpected frequency of about 1.6 Hz.



As a last resort, we could try increasing the sampling rate in the forcing code even more.

ADAMS/Solver – 50 output steps/sec (step size = .020)
External Force - 1000 compute steps/sec (step size = .001)



This is even worse! Not only is the discretely-forced side unstable and responding at a strange 1.4 Hz frequency, the numerics have gotten so bad that this response has coupled into the supposedly good side of the model and is sending it unstable also!

So here we have an apparently simple co-simulation problem, using externally computed discrete forcing, where we can not even get close to the true solution! Increasing the number of simulation steps on the Solver side can artificially excite undesired system harmonics, and increasing the number of steps on the forcing side can send the response unstable.

THE SOLUTION – A LITTLE “GLUE”

There is fortunately a fairly straightforward solution to the problem. This is to add an interpolator/extrapolator interface between ADAMS/Solver and the other code.

First, we must recognize that all digital computer solutions to these kinds of problems are actually discretely computed approximations to the continuous physics. (Let's not consider the area of digital controls for now.) The differences between the various kinds of solution tools we have is mainly in the order of the functions that are used to approximate the true solution between the discrete points where it is computed.

While a finite difference code (and our demonstrator discrete forcer) may make no attempt at all to interpolate between solution points, ADAMS/Solver uses polynomials of varying orders in its predictor/corrector solution to both help the integrator advance and interpolate the response between solution points. We can use this same approach for co-simulation.

The other code wants to take its own preferred time steps, usually much smaller than Solver, and wants to be able to sample the ADAMS response at each of those steps. And since Solver uses mainly variable time step integrators, it is neither reasonable nor efficient to try to force it to use the same small time step as the other code. And as we have shown previously, even if we could do this, it would not entirely fix the co-simulation problem. What we really need is a continuous approximation to the ADAMS results that the other code could sample whenever it wants – an interpolator.

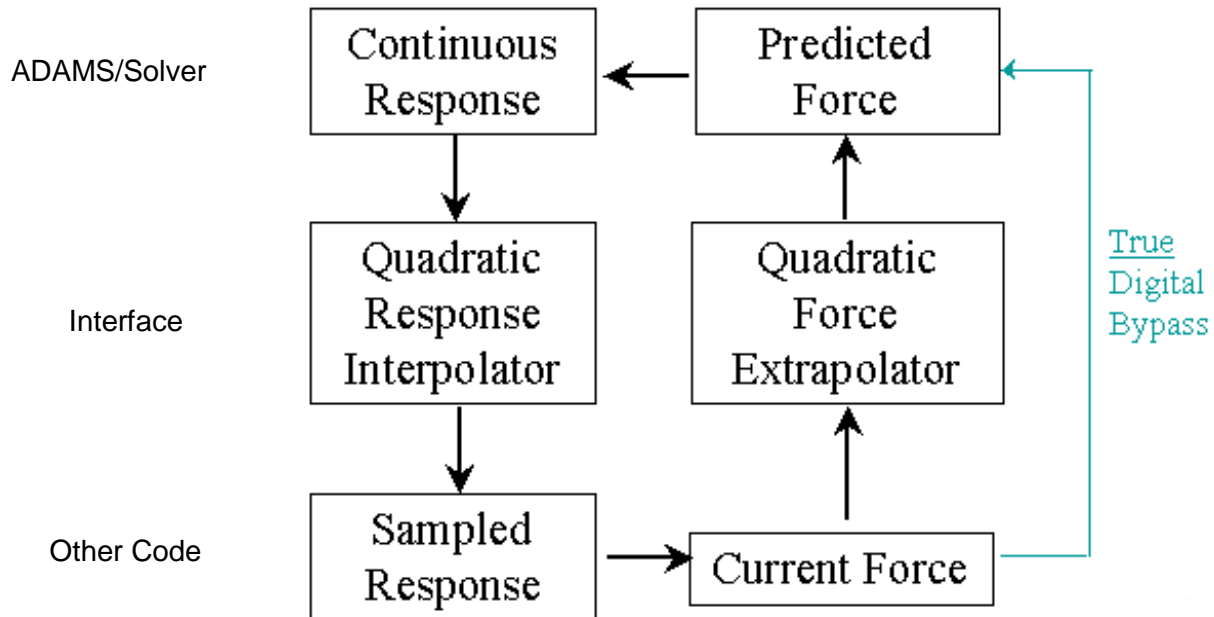
Also, ADAMS/Solver wants to be able to interrogate the other code for its response at any specific time, not only at some fixed interval, and not even always stepping forward in time. The other code typically has only discrete outputs, so it can only respond with those. We might try to make the other code take such tiny steps that it was very close to continuous, but again we have seen earlier that this will not entirely fix the co-simulation problem.

Further, as Solver advances, the predictor needs to guess at the response at future times, but the other code can not provide those. So what we need here is a way to extend the response of the other code into the future – an extrapolator.

So the solution to our problem is to create a small, highly-efficient “glue” routine which connects the two codes during a co-simulation, and can do two-way interpolation & extrapolation on the data that passes between the codes. This actually works extremely well and is described in the following section.

A 2-WAY INTERPOLATING/EXTRAPOLATING INTERFACE

The required functionality from our “glue” routine is shown in the following diagram.



This is typically implemented with the following logic:

1. ADAMS/Solver updates its side of the interface with mechanical response data at each successful integration step.
2. The other code gets an interpolated response from the interface at whatever sampling interval it wants to use.
3. The other code advances until it is within one time step of the Solver simulation time, updating its side of the interface with force data at each step.
4. A/Solver extracts continuous extrapolated forces from interface to advance.

Because the other code never quite catches up to Solver, this is sometimes called a “half-step lead” method.

Note that if the other code is actually simulating a true discrete process, you should not use the extrapolator part of the interface. This is shown in the “bypass” in the above diagram. Similarly, if the other code’s response is not dependent on any ADAMS system states, but only on time, there is no need to use the interpolator part.

The interpolator and extrapolator both use quadratic functions. Using quadratics avoids the “spline buckling” problem that can occur with higher order polynomial, but still gives a much better approximation than simple linear functions. However, the quadratic functions require three data points, so they are more expensive to compute than linear functions and require three steps to get started. It is important, therefore, to do a careful implementation of the interpolator/extrapolator to get the best possible response from the interface.

A single-variate implementation in FORTRAN of such an interface is shown here, based on an analytical solution for the quadratic coefficients and using a rotating stack to minimize memory operations. This can be translated into any desired language, and also can be used as the basis for a multi-variate implementation.

```

SUBROUTINE INTRP2 ( yvals, tvals, reqtim, value )
  DOUBLE PRECISION yvals(3), tvals(3), reqtim, value
  DOUBLE PRECISION a,b,c,denom,y1d23,y2d31,y3d12,d31,d23
C assumption is that  $y = a*t^2 + b*t + c$ 
  d31 = tvals(3)-tvals(1)
  d23 = tvals(2)-tvals(3)
  denom = d31*(tvals(1)*tvals(3) + tvals(2)*(d23-tvals(1)))
  y1d23 = yvals(1)*d23
  y2d31 = yvals(2)*d31
  y3d12 = yvals(3)*(tvals(1)-tvals(2))
  a = ( y1d23 + y2d31 + y3d12 ) / denom
  b = ( (tvals(2)+tvals(3))*y1d23 +
1      (tvals(3)+tvals(1))*y2d31 +
2      (tvals(1)+tvals(2))*y3d12 ) / -denom
  c = ( (tvals(2)*tvals(3))*y1d23 +
1      (tvals(1)*tvals(3))*y2d31 +
2      (tvals(2)*tvals(1))*y3d12 ) / denom
C output
  value = a*reqtim**2+b*reqtim+c
  return
end

```

USING TIMGET

Synchronization between the two programs depends on knowing where each is in the simulation sequence. The ADAMS/Solver utility subroutine TIMGET always returns the time corresponding to the last successful simulation step, and Solver will never backup past this time. By monitoring the result from TIMGET inside a user-written subroutine (SFOSUB, VFOSUB, etc.), we can identify when a successful simulation step has just been made.

This introduces a small complication. Monitoring the result of TIMGET tells us only when the previous step was successful. This means that at each entry to the subroutine, we need to always save all the states that have to be passed to the forcing code, so that when we see the change in the TIMGET result, we can send over the set of states from the previous time the subroutine was called. This is outlined below.

Required Solver-side Functionality

1. Check TIMGET to see if previous step was successful.
(If no, jump to #3.)
2. Link to interface. Update interpolation arrays from saved states array.
3. Update saved states array with current states.
4. Link to interface. Get extrapolated forces.
5. Return forces to model.

INTER-PROCESS COMMUNICATION

Of course, in order to get the co-simulation to work properly, it is required that the two codes and the “glue” routine can talk to each other. There are various ways to get that to happen, depending on the operating system and hardware in use for each code, as well as on whether or not you have access to the internal workings of the other code. On the Solver side, the communication will be most often done through standard user-written subroutines for SFORCE, VFORCE, GFORCE, VARIABLE or DIFF elements.

The most efficient inter-process communication is via a direct subroutine interface. That is, the interpolation/extraction interface and the other code are set up so that they can be called by ADAMS/Solver. In this case, data can be transferred using shared memory, such as Fortran COMMON blocks. On the Windows/NT platform especially, as long as each code has some method of access to system services, this can also be done using dynamic link libraries (DLL's)

Another method for inter-process communication is with *pipes*. Pipes are provided by both the Unix and NT operating systems and are functionally equivalent to shared memory stacks, but are accessed as files. Process blocking and synchronization tools are provided as part of the *pipe* services. Pipes require that all the involved processes be running on the same system, or on similar systems within an NT workgroup. The example which is presented here uses *pipes* as the communication method.

Finally, if the involved processes are running on dissimilar systems or even in different locations, inter-process communication can be implemented using network *sockets*. This is another standard system service provided by both NT and Unix operating systems. Sockets can run on any compatible hardware over any network using the TCP/IP protocols. They are implemented very similarly to *pipes* and also provide process blocking and synchronization tools. A tutorial on using *sockets* for inter-process communication with ADAMS can be found at http://www.members.home.net/a.s.elliott/IPC/IPC_intro.html

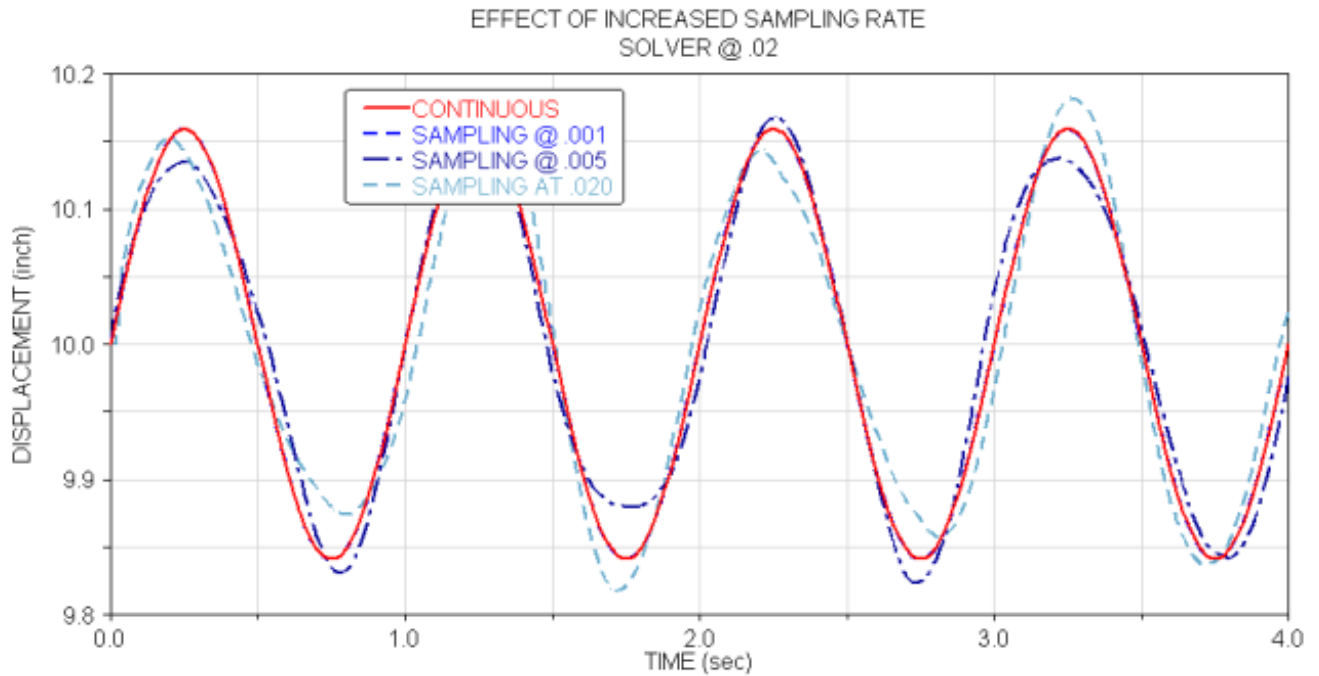
RESULTS

This is one of those few times where you can have it faster, cheaper and better!

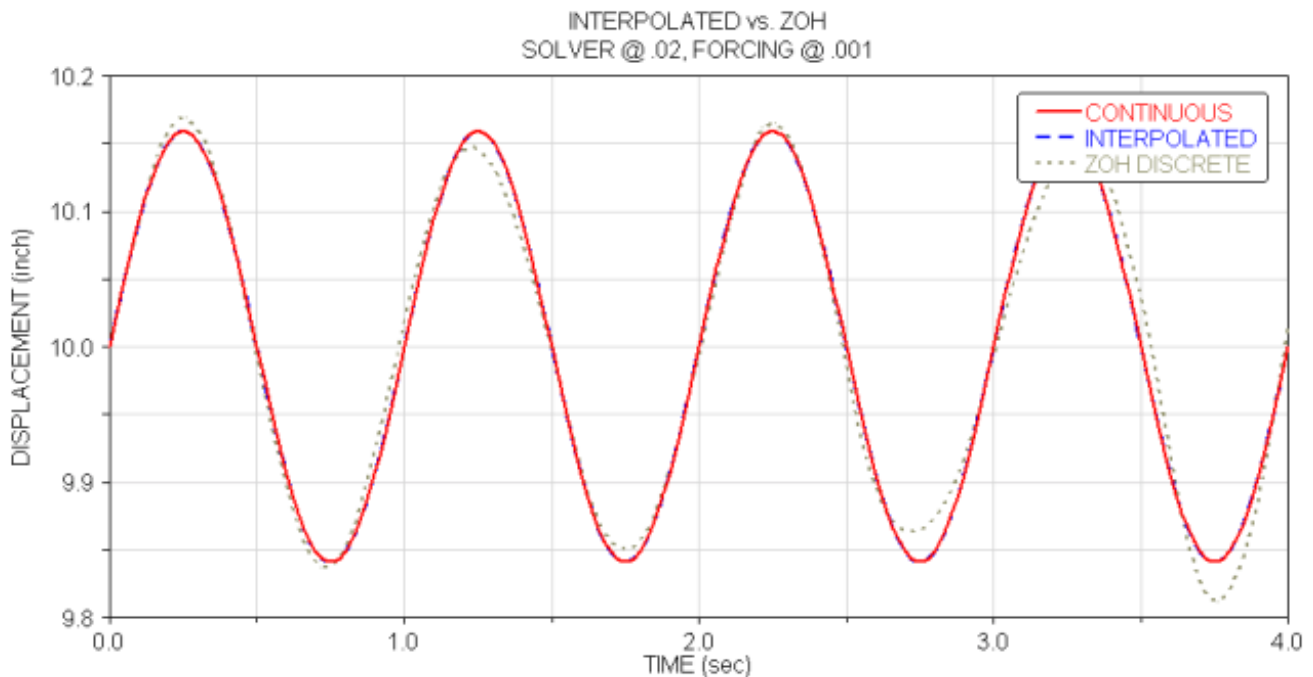
1. The interpolated co-simulation always runs faster than the non-interpolated one. Here are run-time results for the 4-second example simulation using 50 steps/sec in Solver and 1000 Hz sampling in the forcing code, on a 400 MHz NT machine:

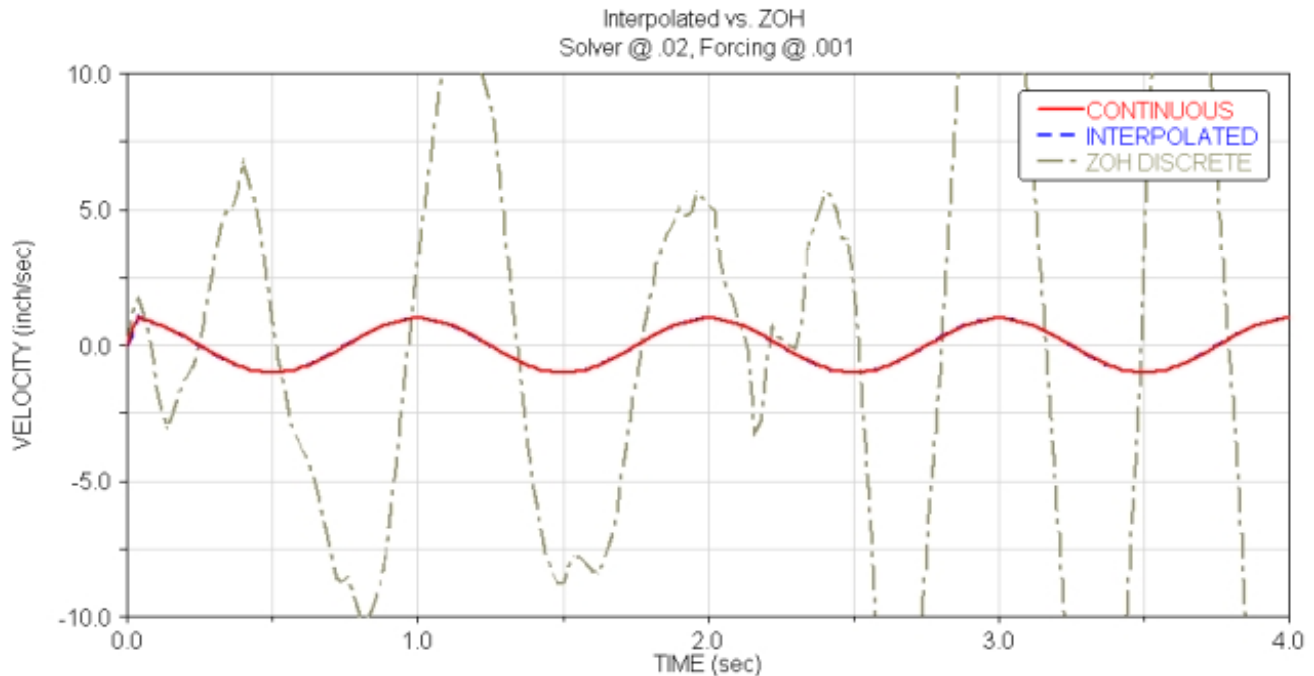
Non-interpolated -	7.91 seconds
Interpolated -	0.73 seconds!

2. It is not necessary to do extensive modifications to the ADAMS model or to the other code. Most of the work is done in the interpolation/extrapolation interface. Depending on your choice for interprocess communication methodology, you can even continue to run your two codes on separate boxes with different operating systems.
3. The interpolated co-simulation converges nicely to the continuous solution as the computational step size (or sampling interval) is decreased in the other code. This is shown in the following plot, where the curve for sampling at .001 seconds is directly below the true continuous solution.




4. The interpolated results are much better than the non-interpolated results, not only in the displacements, but also in the first and second time derivatives. This is shown in the following two plots, where the non-interpolated co-simulations results are referred to as “ZOH”, or zero-order hold. In both plots, the interpolated results are so close to the continuous solution as to be indistinguishable.





SUMMARY


1. Co-simulation is a widely useful technique for joining any type of existing external computations with an ADAMS model.
2. Good implementation of a co-simulation interface can be tricky. A poor implementation can be very slow and even give incorrect results.
3. Using a two-way quadratic interpolation/extrapolation scheme in the interface can greatly improve both co-simulation fidelity and speed.
4. Such an interface can also be used across platforms and across operating systems.
5. The presentation, example models and example code are available on CD or for download on the Internet. Contact the author at aelli@adams.com.




ADAMS

A Highly Efficient, General-Purpose Approach for Co-Simulation with ADAMS

Dr. Andrew Elliott
Technical Specialist
N.A. Professional Services




Mechanical Dynamics



ADAMS

What is Co-simulation? Why use it?

- “Co-simulation” - Connecting some external process to ADAMS/Solver so that it and A/Solver can communicate to each other during the simulation.
- Example Uses
 - ▶ Make use of existing external code for computing forces
 - ◆ Use other external software to control an ADAMS model
 - ◆ Execute hardware-in-the-loop coupled simulations
 - ▶ Do interactive simulation modification



Mechanical Dynamics

ADAMS

Most Important “Considerations”

- ADAMS works in continuous time (and sometimes goes backwards).
- External code may be in discrete time (sampled) and may have multiple or variable sample rates.
- External code may be much slower or much faster than ADAMS/Solver.
- External code may run on separate hardware.
- Communications should allow for maximum possible efficiency for both programs.



ADAMS

Co-Simulation Combinations

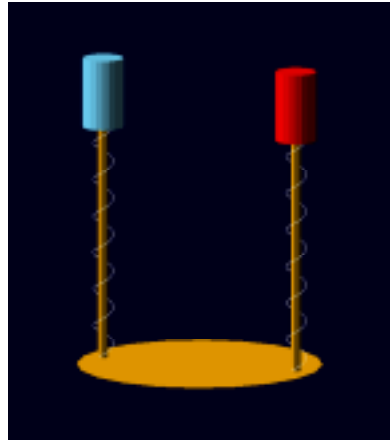
- ADAMS/Solver - always continuous time solution
- Other Code
 - ◆ Continuous response - continuous time solution
 - Example: $L = C_{L\alpha} \alpha(t, \bar{x})$
 - ◆ Continuous response - discrete time solution
 - Example: Lift computed by finite difference CFD code
 - ◆ Discrete response - discrete time solution
 - Example: Digital controller
 - ◆ Combinations (controls + actuators)



ADAMS

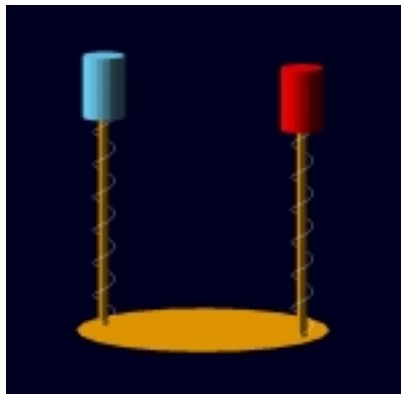
What can go wrong?

- Excessive run times
- Communications bottlenecks
- Synchronization failure
- Numerical “pinging”
- Incompatible error control
- Aliasing
- Artificial instability
- Wrong answers!



ADAMS

Co-Simulation Problems



- Continuous force on one mass. Discrete forces, via co-simulation, on other. 1 Hz Forcing on a mechanical system with $\Omega = 100$ Hz .
- Examine effects of increased resolution in either code.



ADAMS

Continuous Force - SFO1

```
!      adams_view_name='SFO1'  
SFORCE/1, TRANSLATIONAL, I=4, J=2,  
, FUNCTION=50*SIN(2*PI*TIME)
```

Discrete Force - SFO2B

```
!      adams_view_name='SFO2B'  
SFORCE/4, TRANSLATIONAL, I=6, J=7,  
, FUNCTION=USER(0)
```



ADAMS

Discrete Forces - SFO2B

- ADAMS/Solver - SFOSUB
- “Glue” program - IN_EX, with interpolation disabled
- “Other” code - MS Excel, with DLL





ADAMS

Simulation Sequence

<u>Attempt</u>	<u>Steps in 4 sec</u>	<u>Sampling Interval</u>
1	100	.02
2	200	.02
3	200	.002
4	200	.001

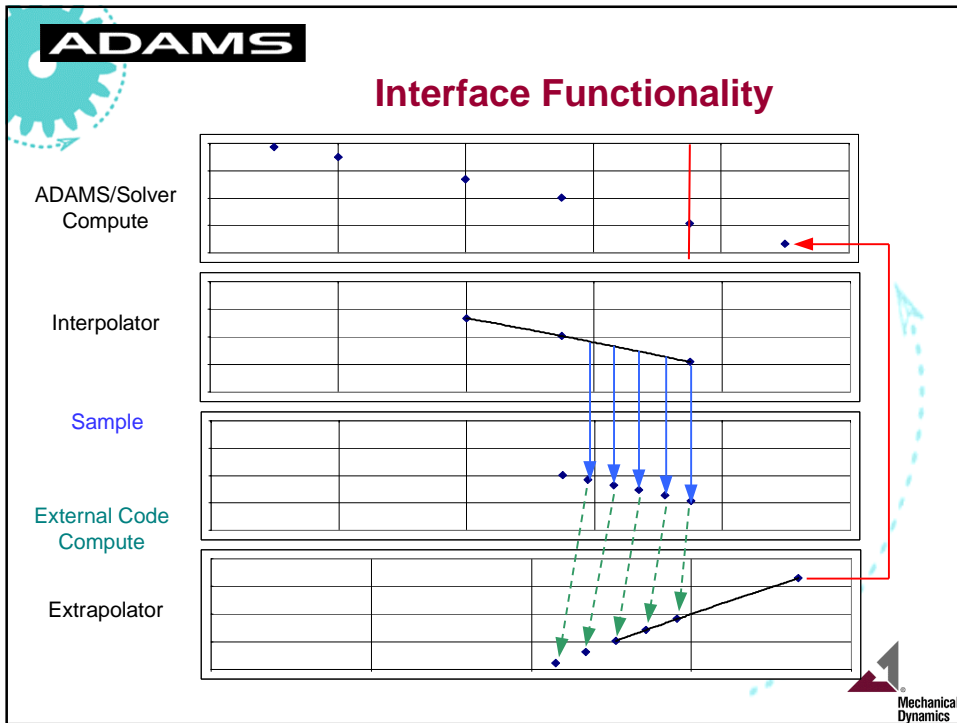
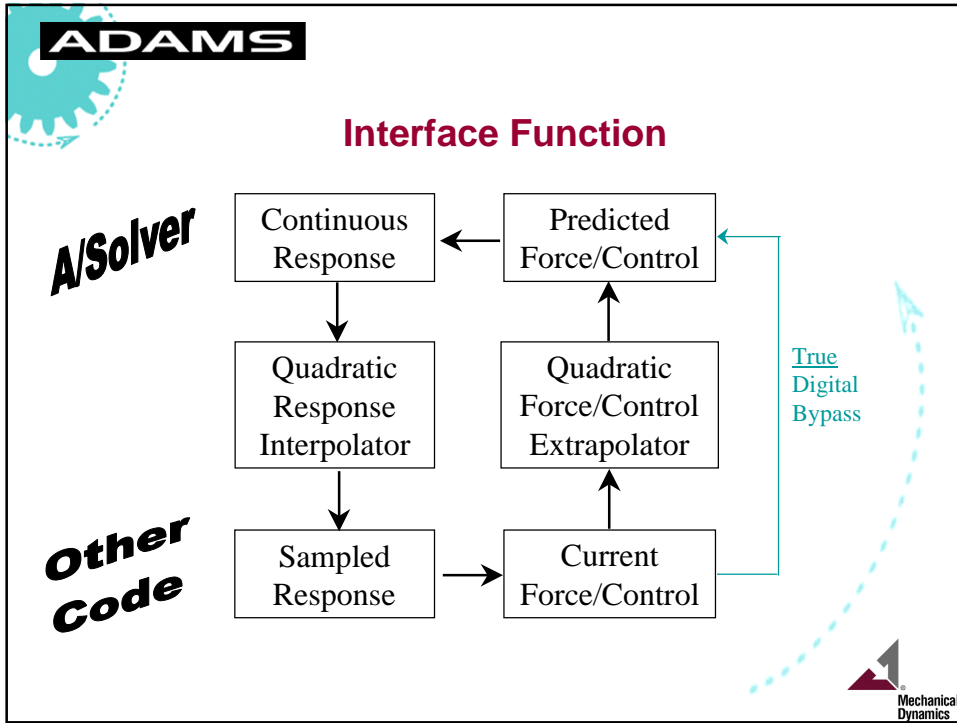


ADAMS

A Solution - Interpolation / Extrapolation

- Create a small, highly-efficient “glue” routine which can do two-way interpolation & extrapolation on the data passed between the codes.
- *Typical* use for discretely-computed external forces:
 1. A/Solver updates interface with response data at each successful integration step.
 2. External code pulls interpolated response from interface at own sampling interval.
 3. External code updates interface with force data at each time step.
 4. A/Solver extracts continuous extrapolated forces from interface.
- Sometimes called “*half-step lead*” method.





ADAMS

Quadratic Interpolation / Extrapolation

- Better approximation than simple linear, but requires 3 data points.
- Avoids “spline buckling” problem with cubics (esp. with uneven data intervals)
- Easy to implement, fast compute time.
- Assume $y = At^2 + Bt + C$
- Analytical solution for A, B, C . (MathCAD)
- Use a rotating stack - no data ordering.



ADAMS

Single-Variable FORTRAN Implementation

```
SUBROUTINE INTRP2 ( yvals, tvals, reqtim, value )
DOUBLE PRECISION yvals(3), tvals(3), reqtim, value
DOUBLE PRECISION a,b,c,denom,y1d23,y2d31,y3d12,d31,d23
C assumption is that y = a*t^2 + b*t + c
d31 = tvals(3)-tvals(1)
d23 = tvals(2)-tvals(3)
denom = d31*(tvals(1)*tvals(3) + tvals(2)*(d23-tvals(1)))
y1d23 = yvals(1)*d23
y2d31 = yvals(2)*d31
y3d12 = yvals(3)*(tvals(1)-tvals(2))
a = ( y1d23 + y2d31 + y3d12 ) / denom
b = ( (tvals(2)+tvals(3))*y1d23 +
1 (tvals(3)+tvals(1))*y2d31 +
2 (tvals(1)+tvals(2))*y3d12 ) / -denom
c = ( (tvals(2)*tvals(3))*y1d23 +
1 (tvals(1)*tvals(3))*y2d31 +
2 (tvals(2)*tvals(1))*y3d12 ) / denom
C output
value = a*reqtim**2+b*reqtim+c
return
end
```



ADAMS

Using TIMGET for Synchronization

- GETSTM returns current simulation time when called. Can change forward and backward. Not very helpful.
- REQSUB is called only at each *output* step. Often too coarse for co-simulation.
- SENSUB is called after each successful integration step and can be/has been used for controlling interface communications.
- TIMGET always returns the time corresponding to the last successful simulation step. Easier and simpler. Solver will not backup past this time. Everything gets done in the xFOSUB or VARSUB.



ADAMS

Update and Save Timing



No	Yes	No	No	Yes	No	Yes	Successful Step?
	No	Yes	No	No	Yes	No	TIMGET Changes?
Yes	Yes	Yes	Yes	Yes	Yes	Yes	Save Step?
No	No	Yes	No	No	Yes	No	Update Arrays?



ADAMS

Inside xFOSUB or VARSUB

Required Functionality:

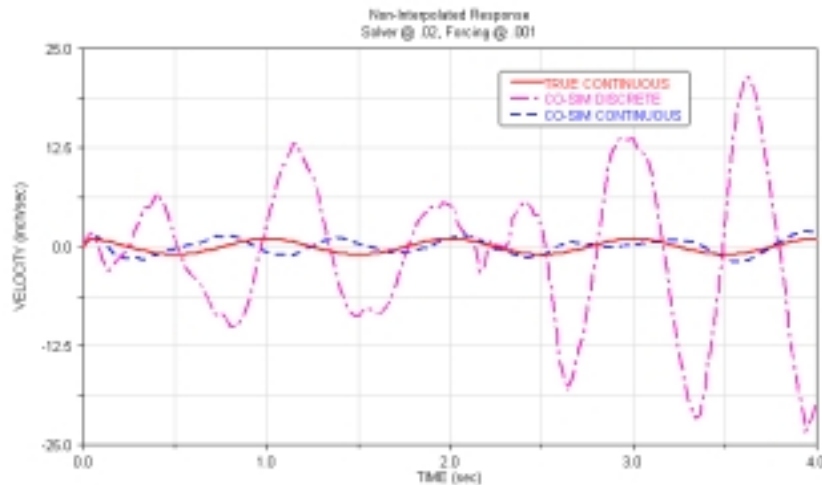
- ◆ Previous step successful?
 - ◆ Update interpolation arrays from saved states
 - ◆ Update saved states
 - ◆ Interpolate states to next sample time
 - ◆ Link to other code; wait.
 - ◆ Update extrapolation arrays.
 - ◆ Extrapolate forces
 - ◆ Return forces to model
- Requires static storage for state interpolation array, [NSTATES+1, 3]
 - Requires static storage for force extrapolation array [NFORCES+1, 3]
 - Requires static storage for synchronization and other time data.
 - Requires static storage for temporary states save array [NSTATES+1, 1]

NO



ADAMS

Review - No Interp, 200 steps, .001 sampling



ADAMS

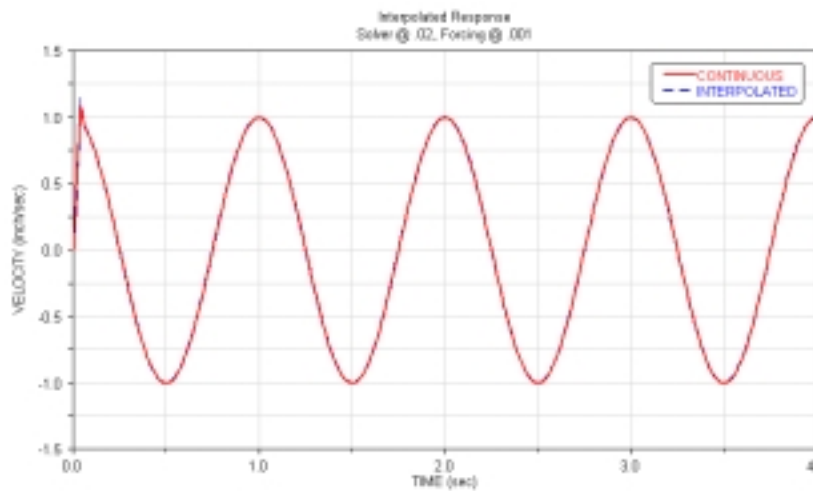
Simulation Sequence

<u>Attempt</u>	<u>Steps in 4 sec</u>	<u>Sampling Interval</u>
1	100	.02
2	200	.02
3	200	.002
4	200	.001



ADAMS

With Interpolation, 200, .001



Inter-Process Communications

- Direct Subroutine Interface
 - ◆ Other code called by Solver, or Solver called by other code
 - ◆ Data passed or contained in COMMON blocks (FORTRAN)
 - ◆ Easiest to implement. Can be done with DLLs.
- Pipes ^{*}
 - ◆ Equivalent to shared memory; accessed as files.
 - ◆ Process blocking and synchronization fairly easy.
 - ◆ Requires single system or like systems
- Network
 - ◆ Very similar to pipes, but OK for dissimilar systems, using *sockets*
 - ◆ See http://www.members.home.net/a.s.elliott/IPC/IPC_intro.html

Summary

- Co-simulation is a very useful technique for joining existing external computations with ADAMS.
- Implementation can be tricky and slow.
- A quadratic interpolation/extrapolation interface and greatly improve co-simulation fidelity and speed.
- Such an interface can also be used across platforms and across operating systems.
- Presentation and example code available on CD or for download.
- Contact for author: aelli@adams.com