Note:  and .

**Initialization**

At time , an input-output solve from Step 3 is performed where  and  are replaced with  and  is replaced with .

**Time Stepping**

Data known at :

[[1]](#footnote-1)

for all .

**Step 1: Extrapolate Inputs**

Let  (correction counter;  represents the prediction) and  (input-output-solve iteration counter) and extrapolate all inputs to yield predicted input values at , i.e.,

,

for all .

**Step 2: Advance States**

Advance the state of all partitions to yield state and constraint values at , i.e.,

,

for all .

If  needs  for , these can be accessed by using an interpolation of inputs, i.e.,  (but this is not needed if  is explicit). If  is explicit for all partitions, the correction will not improve anything (but the prediction is adequate).

If a module has an intrinsic time step less (by an integer multiple) than the global time step, ,  would be called repeatedly in succession to advance the states to .

**Step 3: Input-Output Solve**

*Option 1: Solve for consistent inputs and outputs, which is required when  has direct feedthrough in modules coupled together.*

Solve for consistent inputs and outputs based on the states at  using a root-finding algorithm, i.e., solve

, or equivalently, 

for , which also gives .  represents a call to each module’s . For Newton’s method this would be:

.

The Jacobian  can be computed directly if all Jacobians are available to the glue code,

,

or numerically,

,

where  is a vector of zeros with the th element equal to 1 and  is a perturbation of the th input.

*Option 2: Solve inputs only based on the current outputs, which is an approach much faster than Option 1 and can be used when the modules coupled together do not have direct feedthrough.*

Solve for inputs from current outputs based on the states and current inputs at  using a root-finding algorithm, i.e., solve

, or equivalently, 

for , which also gives .  represents a call to each module’s . This may be trivial to solve.[[2]](#footnote-2) Otherwise, for Newton’s method this would be,

,

which only requires  to be called once per module.

Note: If Options 1 and 2 are used together (some module couplings using Option 1 and some using Option 2), then the modules coupled using Option 2 should be solved first. In this case,  should be reset to 0 at the completion of Option 2, before starting Option 1.

**Step 4: Correct or Save**

If , let  for all , , and , perform a correction by repeating Steps 2-3.

If , save all the final variables,

,

for all , which completes solution advancement to time .

To advance to the next step, set  and repeat Steps 1-4.

For modules with time steps larger than the coupling step (L = 1 for identical, L = 2 for two times, etc.), the following algorithm is suggested.

Note:

\*Corrections would be expensive (and may counter the benefit of a larger time step) because they would require N\*(C+1) calls to \_UpdateStates, where N is the number of glue code time steps and C is the number of corrections (C=0 represents a prediction step only). It may be preferable to not correct modules with a time step larger than the glue code time step, which then only requires N/L calls to \_UpdateStates. That is, adding corrections requires L\*(C+1) times as many calls to \_UpdateStates. Incidentally, this is the same number of calls to \_UpdateStates that is required with time-step subcycling (where L is the integer number of module steps within the glue code step), but each step of the later is of a smaller time step.

\*Option 2 would work poorly for modules with strong direct feedthrough.

1. Extrapolate inputs (likely based on inputs stored at the module’s time step)



1. Advance states



1. Input-Output solve

Option 2: Solve for y\_n+L, interpolate to find y\_n+1, then solve for u\_n+1:

 



Option 1:

Solve for u and y at substeps t\_n+1 where t\_n < t\_n+1 <= t\_n+L by only calling CalcOutput at n+L:



1. Correct (go back to 2) or Save
1.  may also be saved in  if a module uses a multi-step method. [↑](#footnote-ref-1)
2. For a trivial , such as , where , this reduces to . [↑](#footnote-ref-2)