Automated Solution Convergence with Adaptive Time Stepping

Dr. C.D. Clark III

January 18, 2015

0.1 Introduction

This is document describes a method for adjusting the time step used by a numerical heat solver in order meet some requirement numerical error.

0.2 Adaptive Time Stepping

BTEC allows for the use of an "Adaptive Time Step", which will exponentially increase the time step used by the heat solver in order to speed up simulation times. The time step is initially set to some minimum value and is then allowed to "grow" by some scaling factor until it reaches some maximum time step. Both the minimum and maximum time steps must be specified by the user. In an attempt to minimize numerical errors, the method considers times at which the temperature will be expected to change rapidly (such as when a laser that was off is turned on) and reset the time step it its minimum value and lets it grow again.

While this method does allow for shorter run times, it does not grantee any level of accuracy. If the user wants to make sure that the temperature solution produced by BTEC has "converged", meaning that the numerical error due to truncation is small, then they must run the same simulation with two different time step limits and compare the solutions.

0.2.1 Automating Convergence

The idea of convergence is simple. If two simulations, identical in every way other than the time step used, produce significantly different answers, then the solution has not converged. We can say that the solution based on the larger of the two time steps is inaccurate, but we cannot say whether the same is true for the solution based on the short time step. In order to determine if the short time step solution is accurate, we would rerun the simulation with an even shorter time step and compare. If these two solutions are significantly different, then again, the solution has not converged. We continue doing this until we get two solutions that are "the same".

This process of running the same simulation with smaller and smaller time steps until two solutions that agree can be automated. It is simple enough to define a metric for which two solutions can be compared and a decision on whether or not they are "the same" can be made. The most obvious metric would be to compare the temperature given by each simulation at every point in the grid and determine the percent difference between the two. If this difference is above some threshold, the solutions are said to differ, otherwise they are considered the same.

Our method for automating the convergence test is similar to what a user would to manually, as described above, but we will not compare solutions from two separate simulations. Instead, we will check for convergence at each step forward by the heat solver. The solver will first try to step forward by the time step specified by some external means (either fixed or stretching). In addition, it will compute the same temperature solution by taking two half steps. These two solutions are then compared, and if the are "the same" (based on the metric described below), the solution is used. If they differ, then time step is cut in half, and the process is repeated. This is done until the desired level of convergence is reached, or until some absolute minimum time step is reached.

Metric

We use percent difference as a metric for determining if two solutions are different. There are multiple ways of using percent error for this task. A common method would be to calculate the *total* percent difference,

or the *average* percent difference by adding up the percent difference at each node (actually, you would add up their absolute values or square). Because this piece of the simulation is critical in terms of impact on runtime, we will instead just consider the percent difference at each node. This way, we do not need to keep a running total of the errors, and we can break out of the check as soon as a node violating the requirement is found.

Let $T_{i,j}^n$ be the temperature at a specific node with z and r indexes i and j, at some time which corresponds to n steps forward by the thermal solver. The job of the thermal solver then is to determine $T_{i,j}^{n+1}$ from $T_{i,j}^n$. Let this operation be denotes an operator acting on the current thermal solution,

$$T_{i,j}^{n+1} = \hat{U}(\Delta t) T_{i,j}^n,$$

where $\hat{U}(\Delta t)$ is the time evolution operator that steps the solution forward by a time step Δt. Our algorithm will consist of computing two different temperature solutions,

$$\begin{aligned} T_{i,j}^{n+1} &= \hat{U}(\Delta t) T_{i,j}^n, \\ T_{i,j}'^{n+1} &= \hat{U}(\Delta t/2)\hat{U}(\Delta t/2) T_{i,j}^n = \hat{U}(\Delta t/2)^2 T_{i,j}^n. \end{aligned}$$

Let α denote our "tolerance" for the percent difference, that is, we require the absolute value of the percent difference at each node to be less than α,

$$\left| \frac{\hat{U}(\Delta t) T_{i,j}^n - \hat{U}(\Delta t/2)^2 T_{i,j}^n}{\hat{U}(\Delta t) T_{i,j}^n + \hat{U}(\Delta t/2)^2 T_{i,j}^n} \right| < \alpha. \tag{1}$$

We accept a solution if and only if this condition is true for all nodes. This parameter α will specified by the user and in practice will most likely not have to be adjusted once a "good" value is found.

Optimization

As mentioned above, this piece of the simulation is critical for run times. In order to reduce the run time required for this automated convergence algorithm, we will employ a few optimizations.

1. A solution will be considered "bad" on the first instance that Condition 1 is broken. We will not attempt to measure the "badness" by continuing to calculate the percent difference at other nodes after the first invalid node has been found.
2. Time steps we be halved until a convergent solution is found. This will allow us to use the temperature solution found by one application of $\hat{U}(\Delta t/2)$ as the "full step" solution to compare against for the next iteration of our algorithm. This will require 4 separate copies of the thermal grid to be maintained. One for the original temperature solution, one for the solution after a full time step Δt obtained by both the application of $\hat{U}(\Delta t)$ and $\hat{U}(\Delta t/2)^2$, and one for the solution obtained by a single application of $\hat{U}(\Delta t/2)$. This will require 4 times as much memory as a non-automated time stepping scheme, but only 33% more memory than an algorithm that would not store the half step solution. We are trading memory efficiency for CPU efficiency.

If we where using an explicit method, we could take advantage of the fact that the system is linear. An explicit method will determine the time derivative of the temperature from the spatial distribution of the temperature *at the current time step.* solution based on only the previous solution, it would be the case that

$$\frac{T^{n+1} - T^n}{\Delta t} = \hat{D} T^n. \tag{2}$$

The right-hand side here is constant, so the temperature difference calculated for a time step $\Delta t/2$ will be half of the temperature difference computed for a time step Δt. Therefore, the temperature at a half step could be computed directly from the temperature at a full step. With an implicit method, this is not true. And even for the alternating direction implicit method used by BTEC we would still have,

$$\frac{T^{n+1} - T^n}{\Delta t} = \hat{D}T^n + \hat{D}T^{n+1}. \tag{3}$$

So, the solution depends on the time step used.

Algorithm

The algorithm for stepping a temperature solution forward in time is described below. For the sake of brevity, let different temperature solutions obtained during the process be denoted as follows.

- $T(0)$ - the initial temperature solution
- $T(\Delta t)$ - the temperature solution at a time Δt in the future obtained by a single application of the heat solver
- $T^2(\Delta t)$ - the temperature solution at a time Δt in the future obtained by a two applications of the heat solver based on half time steps.

The algorithm is as follows:

1. Get desired time step, Δt, from time stepper instance (this may be fixed or stretched)
2. Determine $T(\Delta t)$, keeping a copy of $T(0)$.
3. Check if the auto convergence flag is set.
   (a) If Yes
      i. Determine $T(\Delta t/2)$ and store.
      ii. Determine $T^2(\Delta t)$.
      iii. Begin looping through all nodes in thermal grid, computing the percent difference between $T(\Delta t)$ and $T^2(\Delta t)$.
      iv. Invalid node found
         A. Exit loop immediately.
         B. Replace $T(\Delta t)$ with $T(\Delta t/2)$
         C. If $\Delta t/2$ is less than the minimum time step (set by user)
            - continue to Step 3(b)i
         D. If $\Delta t/2$ is greater than minimum time step (set by user)
            - set $\Delta t = \Delta t/2$
            - continue to Step 3(a)i
      v. All nodes valid
         A. Replace $T(\Delta t)$ with $T^2(\Delta t)$ (this should be the more accurate solution of the two).
         B. Throw away $T(\Delta t/2)$ and $T(\Delta t)$.
         C. Continue to 3(b)i
   (b) If No
      i. Throw away original temperature solution.
      ii. Return to caller.