Solution to Burger's Equation (Inviscid)

Ian William Hoppock

April 11, 2017

1 General Remarks and Documentation

The program is written in C, and it solves the inviscid Burger's equation

$$u_t + uu_x = 0$$

such that $x \in [0, 1]$ with periodic boundary conditions. The initial condition is $u(x, 0) = \sin 2\pi x$. The code runs the first, second, and third Adams-Bashforth methods in time and a central finite difference for space.

Rewriting our equation, we have $u_t = -uu_x$. Thus, the standard central difference

$$u_x = \frac{u_{j+1} - u_{j-1}}{2dx}$$

becomes

$$u_t = \frac{-u_j^n(u_{j+1}^n - u_{j-1}^n)}{2dx}.$$

We are now come to the following Adams-Bashforth methods for the inviscid Burger's equation:

1. One-Step/Backward Euler:

$$u_j^{n+1} = u_j^n + dt\left(\frac{-u_j^n(u_{j+1}^n - u_{j-1}^n)}{2dx}\right)$$

2. Two-Step:

$$u_j^{n+1} = u_j^n + \frac{dt}{2}\left(\frac{-3u_j^n(u_{j+1}^n - u_{j-1}^n)}{2dx} + \frac{u_j^{n-1}(u_{j+1}^{n-1} - u_{j-1}^{n-1})}{2dx}\right)$$

3. Three-Step:

$$u_j^{n+1} = u_j^n + \frac{dt}{12}\left(\frac{-23u_j^n(u_{j+1}^n - u_{j-1}^n)}{2dx} + \frac{16u_j^{n-1}(u_{j+1}^{n-1} - u_{j-1}^{n-1})}{2dx} - \frac{5u_j^{n-2}(u_{j+1}^{n-2} - u_{j-1}^{n-2})}{2dx}\right).$$

Nota Bene: these are simplified with respect to signs only. Moreover, the initialising steps for the second and third Adams-Bashforth methods must be worked out using the simpler ones. I.e., the first step of the Two-Step and Three-Step methods use the One-Step/Euler, and the second step of the Three-Step method uses the Two-Step method.

The simulation is compiled by calling some variation of `gcc ab.c vector.c` while having `linear_algebra.h` in the same directory. Under the function `pick_your_adams`, the user may indeed pick the Adams-Bashforth method one wishes to call. Indeed, the output will work just fine if one uncomments all Adams-Bashforth methods in the `pick_your_adams` function. In the `main` function one may set the resolutions for time and space, and immediately above one may modify the initial condition, should one be so inclined.

The simulation follows the following order

`main-->burgers-->pick_your_adams-->ab_1`, `ab_2`, or `ab_3`.

The functions `rhs` and `vector_add` are used repeatedly in the `ab_X`. Autotools are not used for this, as the code was written in one sitting and is quite easy to follow.

Remember: the user must compile this with `vector.c`.

2 Exact Solution

The thorough treatment is omitted by instructions. However, briefly, we have the characteristic equations as a system of ordinary differential equations such that

$$\frac{\mathrm{d}x}{\mathrm{d}t} = u, \ \frac{\mathrm{d}u}{\mathrm{d}t} = 0.$$

The solutions to the characteristic equations are

$$u = C_1, \ x = ut + C_2.$$

However, C_1 is a function of C_2, thus

$$u = C_1(C_2) = C_1(x - ut).$$

Now, from the initial condition, we have

$$u(x, 0) = C_1(x) = \sin 2\pi x.$$

Making this substitution, we have $u = \sin 2\pi x - ut$ and rearranging gives

$$u(x, t) = \frac{\sin 2\pi x}{1 + t}.$$

3 Results

We see, as expected, the sine wave being transformed into a 'saw tooth'. The following diagrams show the functionality of of the code with some interesting twists. Please consider the caption to each, which will tell the spatial resolution J, time step N, and temporal resolution dt. The y-axis is the amplitude of the sine curve, and the x-axis is the position, as expected.

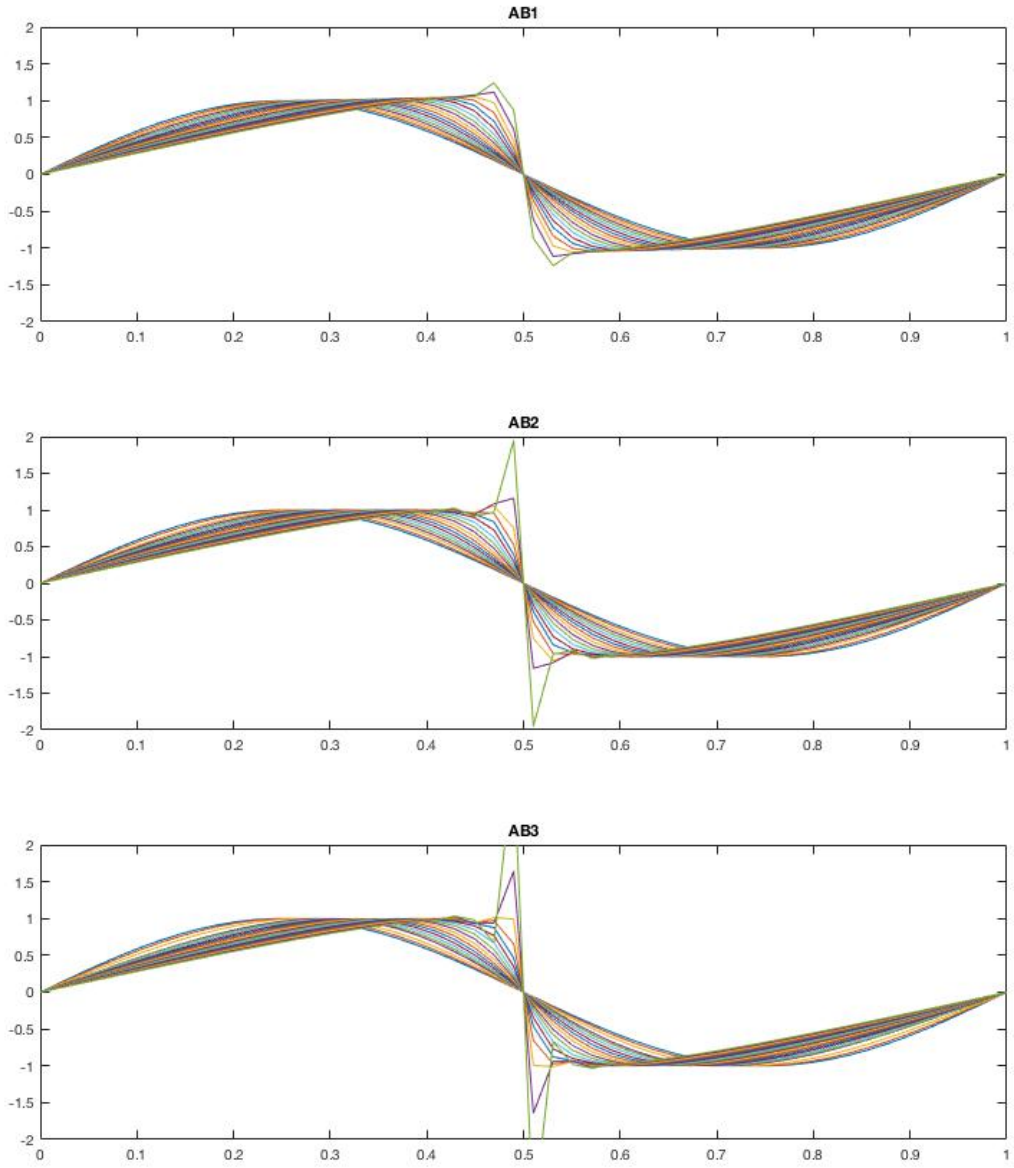

Figure 1: $J = 50$, $N = 19$, $dt = 0.01$. Notes: This behaves exactly as we expect. We start to see damping at just slightly earlier time steps for AB1 than AB2 than AB3. However, AB3 blows up far more spectacularly.

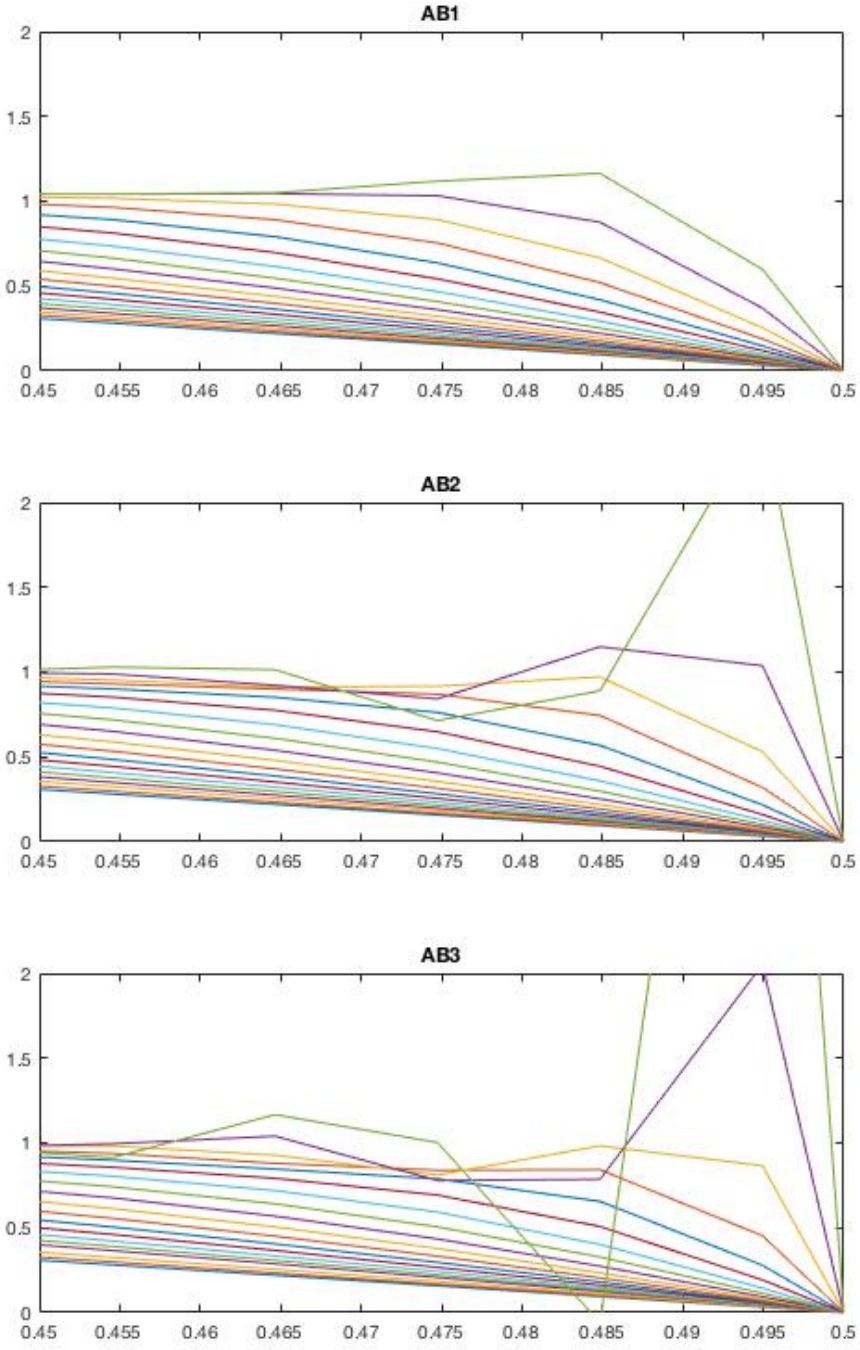

Figure 2: $J = 100$, $N = 19$, $dt = 0.01$. Notes: Double spatially resolved as the previous figure to accommodate the zoom, which shows just how these algorithms break down on approach to the cusp.

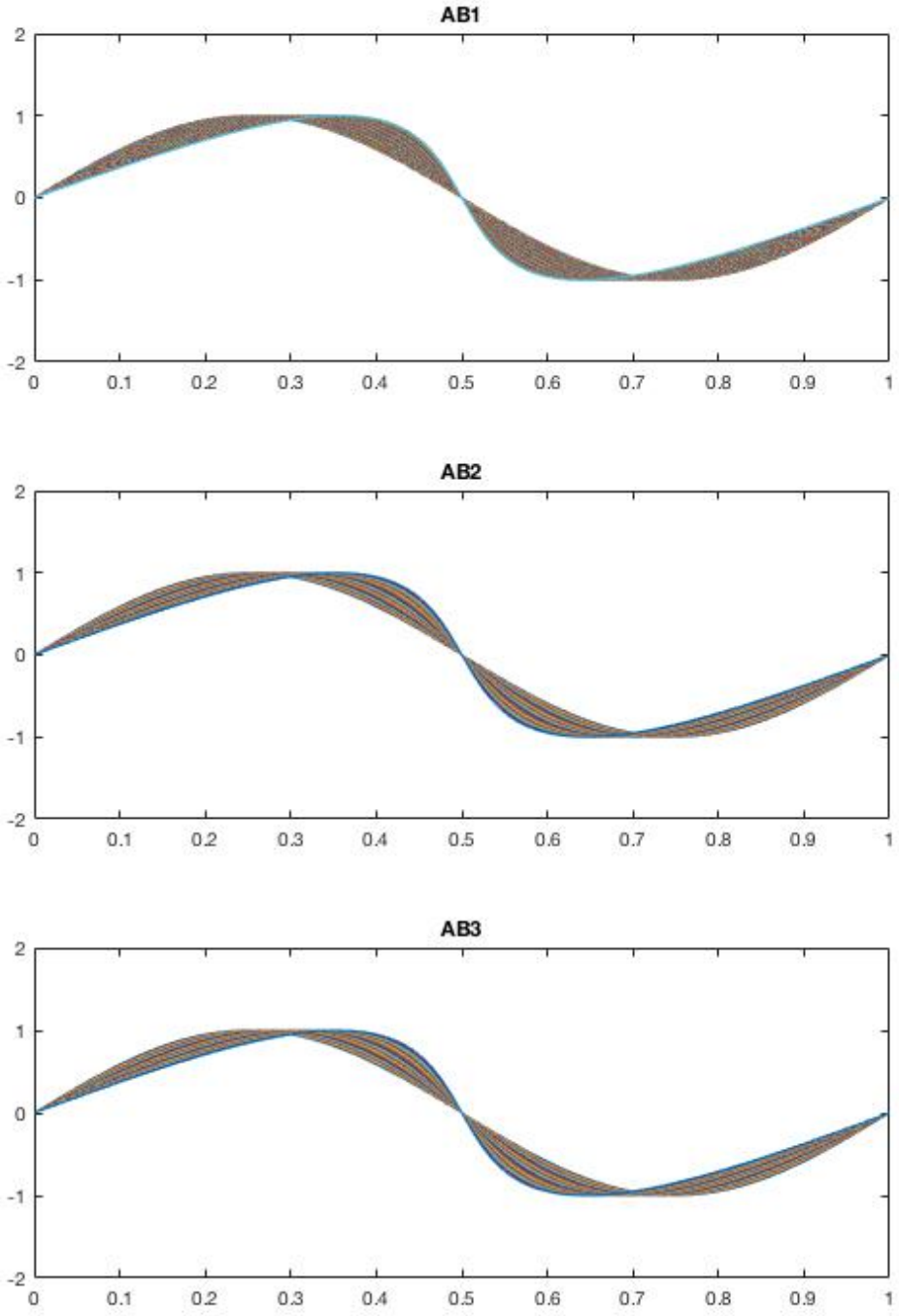

Figure 3: $J = 100$, $N = 110$, $dt = 0.001$. Notes: A standard evolution graphed before the cusp. All amplitudes are the same. Moreover, we see that the progression is relatively similar for each algorithm, i.e., a sign all three are correct.

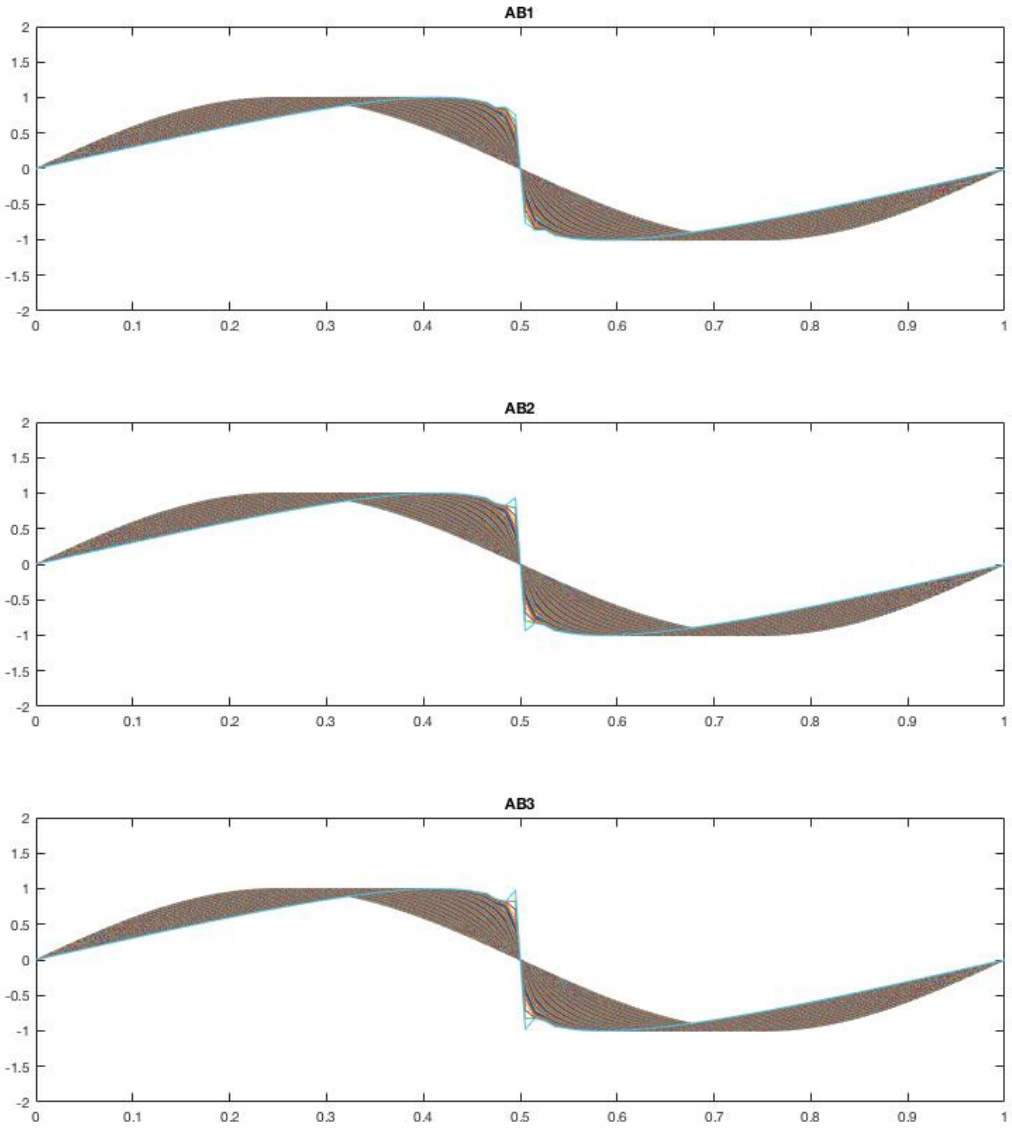

Figure 4: $J = 100$, $N = 165$, $dt = 0.001$. Notes: The same scenario as the prior figure, allowed to continue till cusp formation. Indeed, it is clear by the many squiggles that AB1 starts to fail before the later ones.

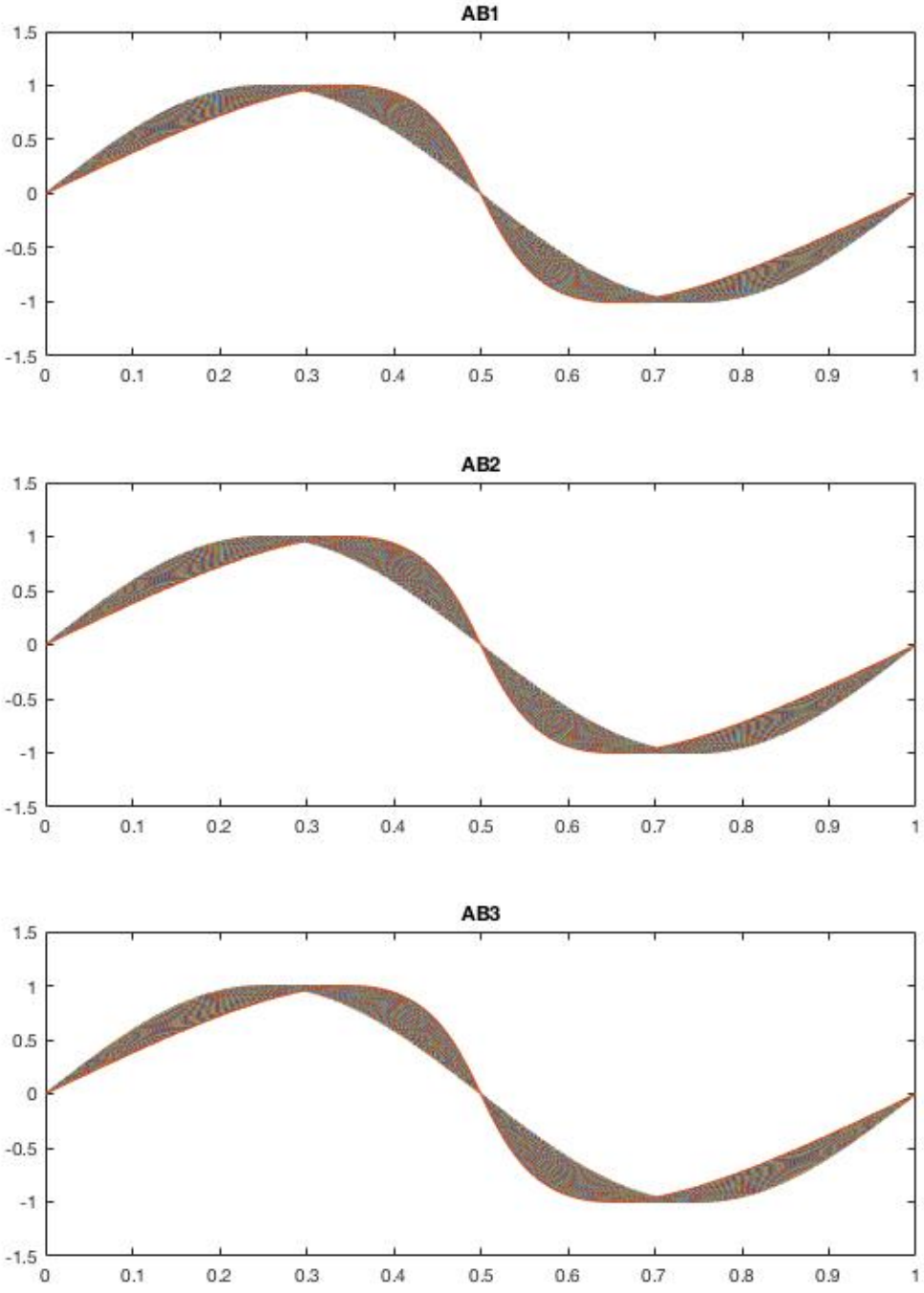

Figure 5: $J = 500$, $N = 100$, $dt = 0.001$. Notes: Simple increase of spatial resolution to test the algorithms capabilities.

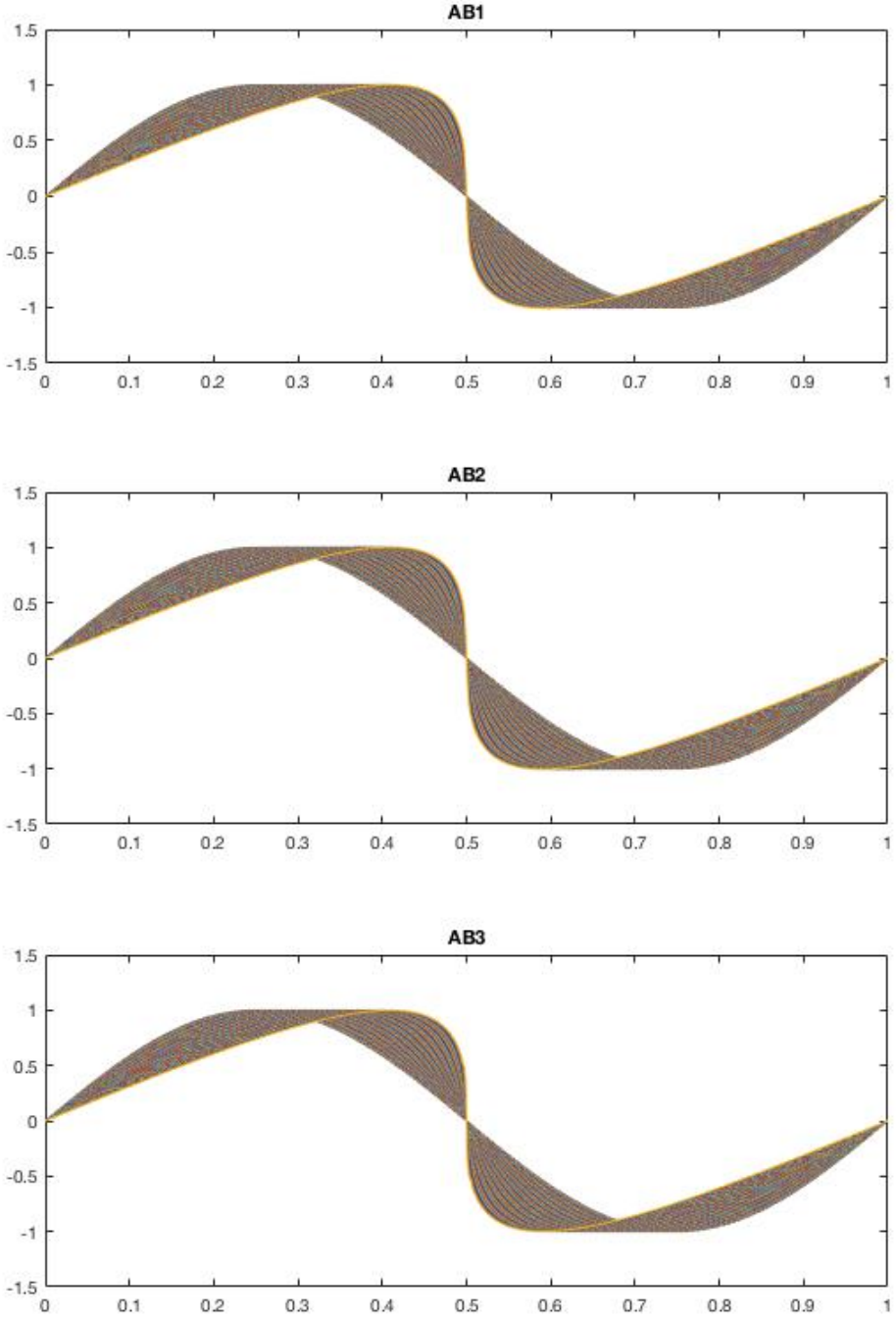

Figure 6: $J = 500$, $N = 160$, $dt = 0.001$. Notes: Graphed just prior to the break down.

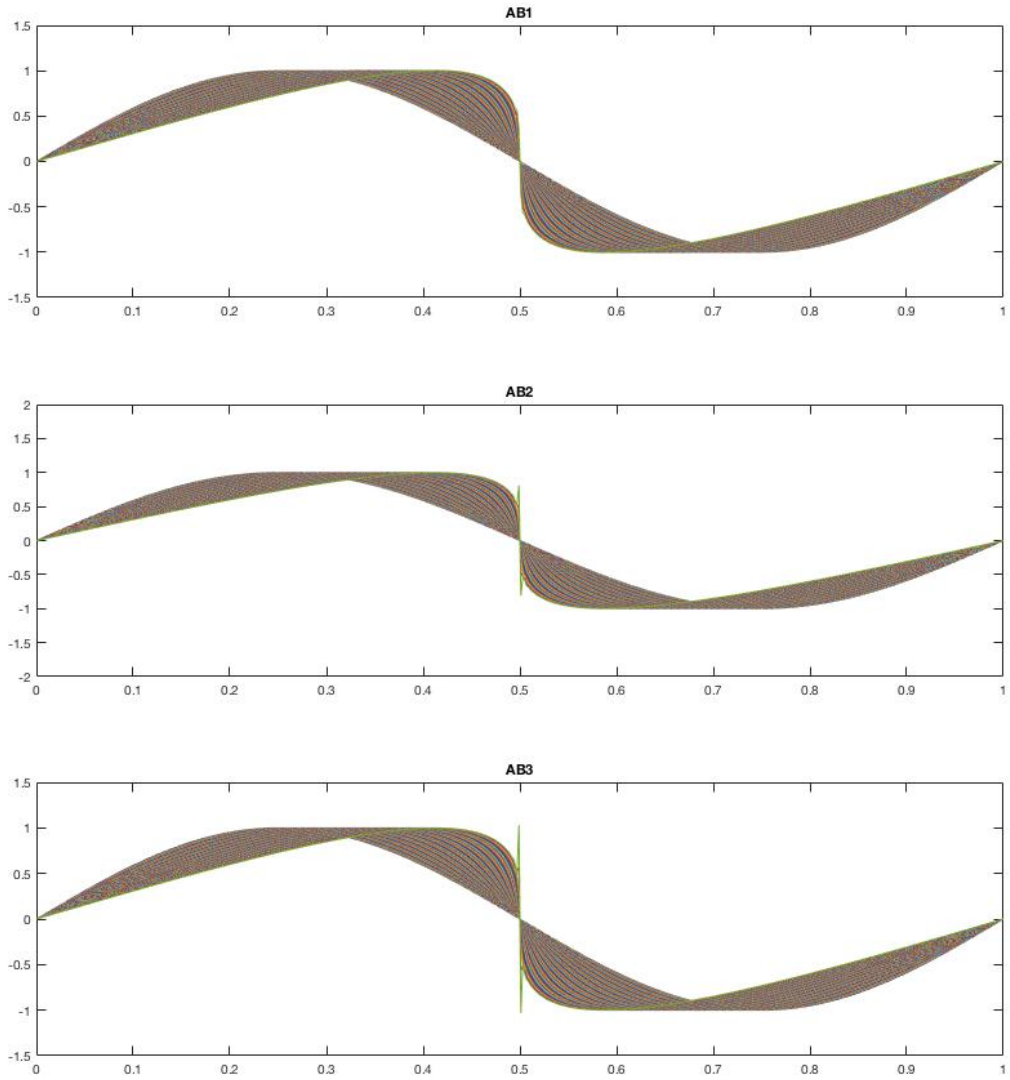

Figure 7: $J = 500$, $N = 164$, $dt = 0.001$. Notes: Indeed cusp formation is more stable with higher spatial resolution. It is less 'pointy' and substantially more resolved.

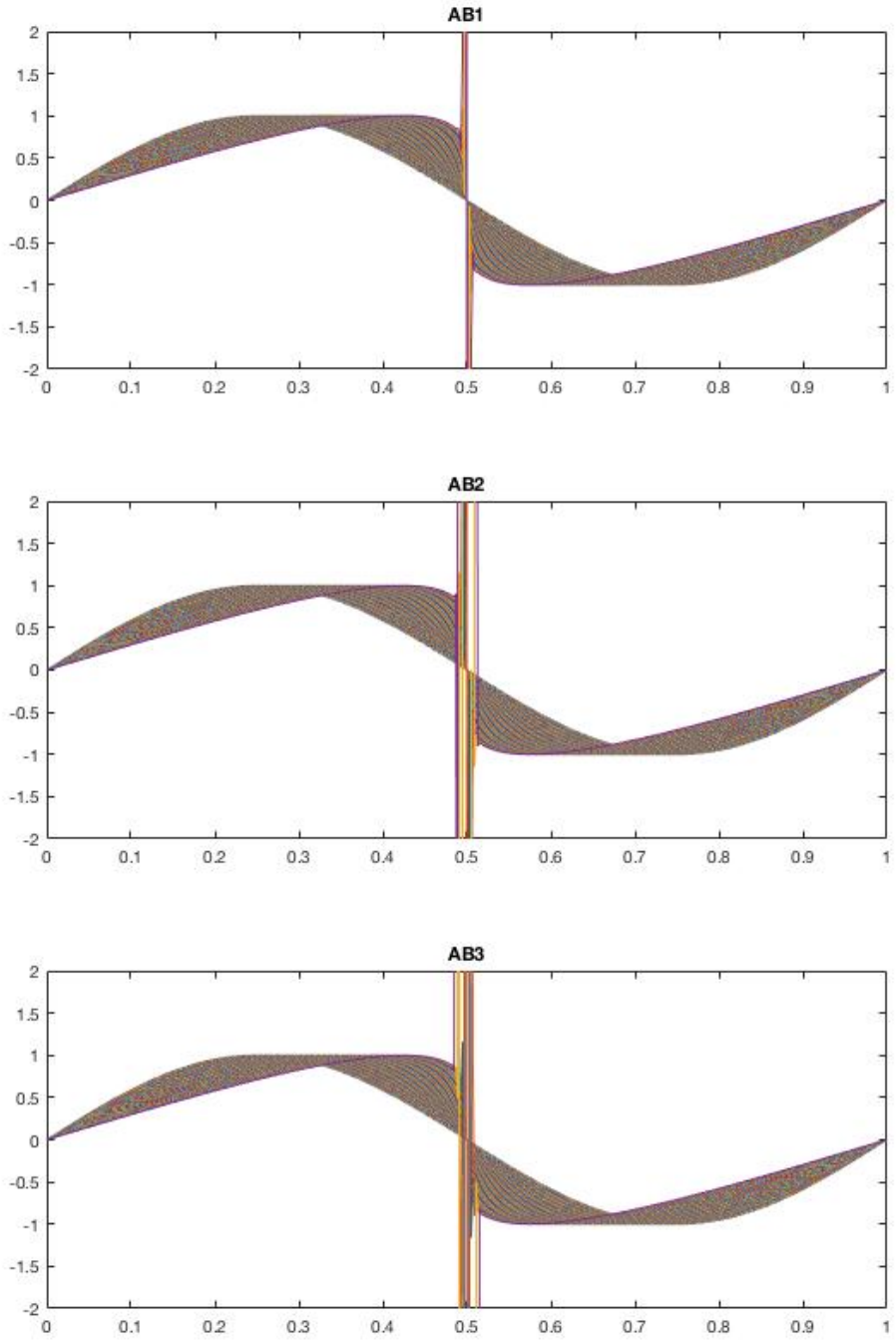

Figure 8: $J = 500$, $N = 175$, $dt = 0.001$. Notes: Graphed at about 10 time steps passed the initial break down. Clearly all three graphs fail spectacularly. Indeed, the upper y-axis limit is on the order of 10^{324}.

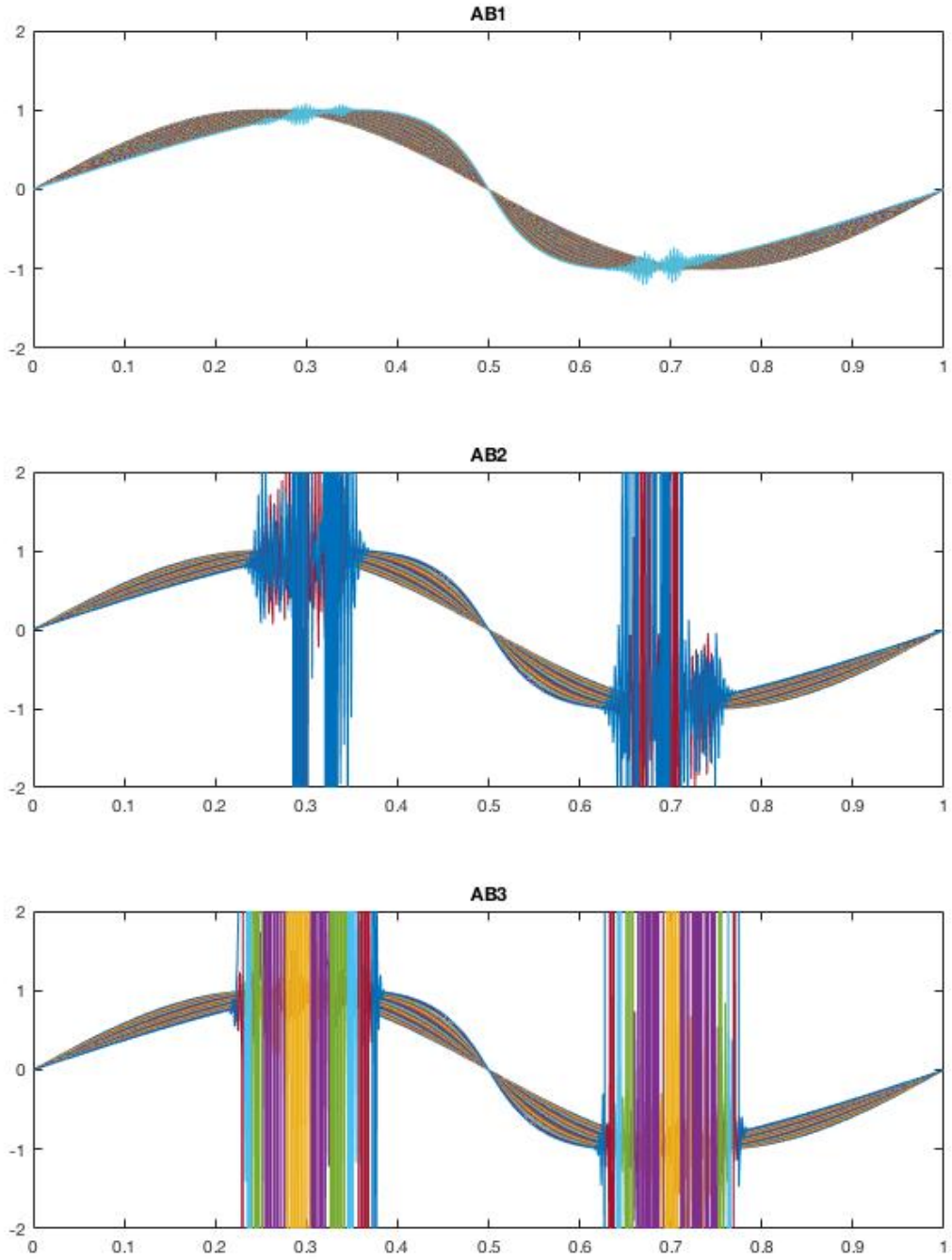

Figure 9: $J = 1000$, $N = 110$, $dt = 0.001$. Notes: An unexpected result, prompting further investigation in the next graphs. We are not expecting a breakdown here, yet it truly does so.

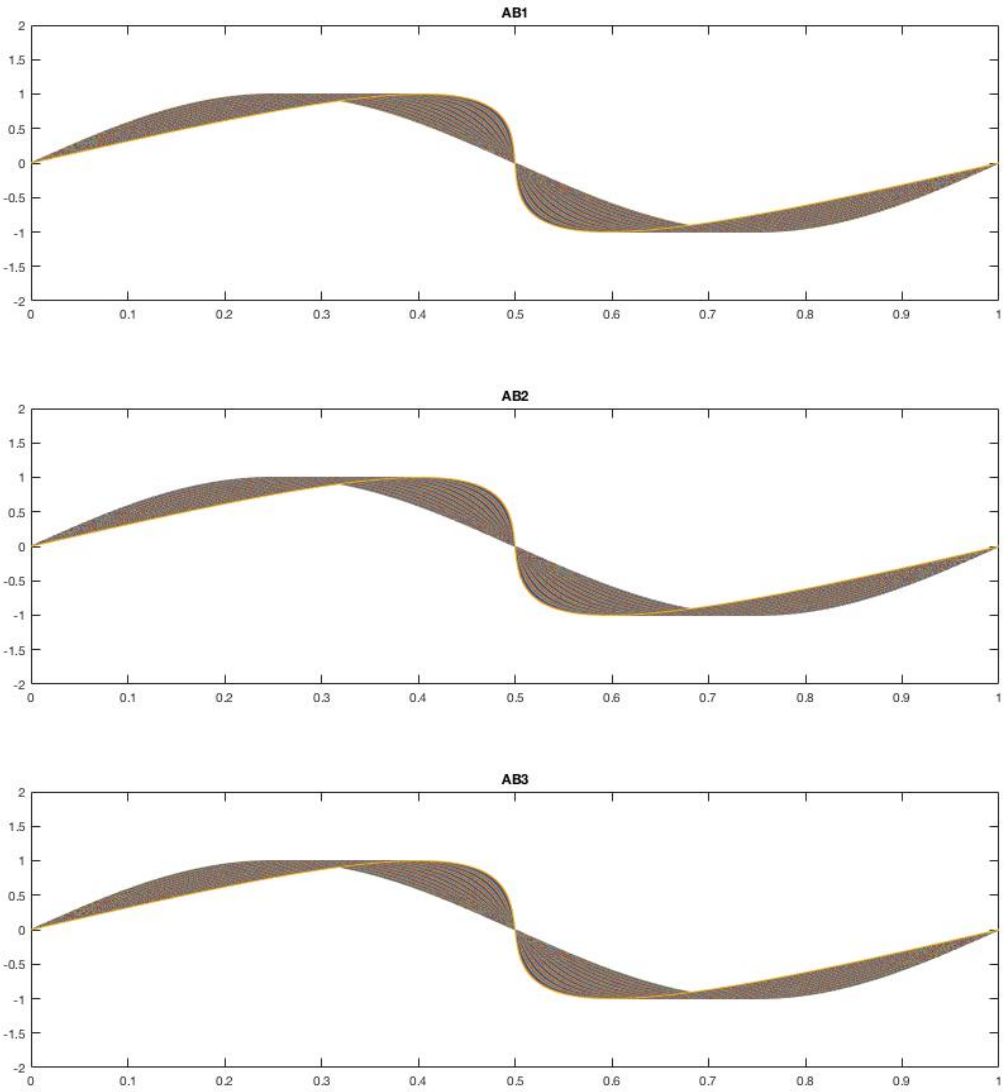

Figure 10: $J = 1000$, $N = 1600$, $dt = 0.0001$. Notes: We resolve this by reducing our dt by an order of magnitude and substantially increasing the number of time steps.

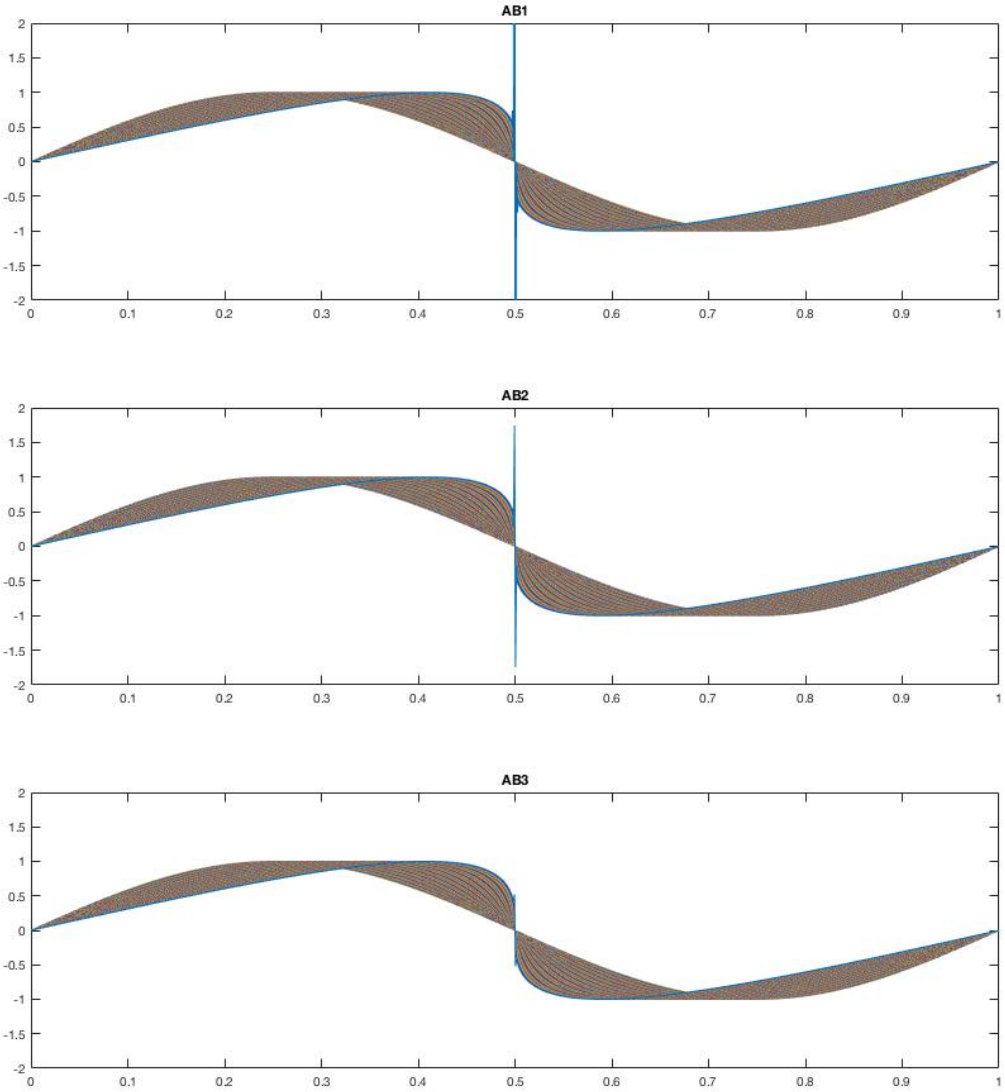

Figure 11: $J = 1000$, $N = 1700$, $dt = 0.0001$. Notes: Allowing this scenario to continue until the breakdown, we see it doing so normally, again, with AB1 failing before AB2 before AB3.

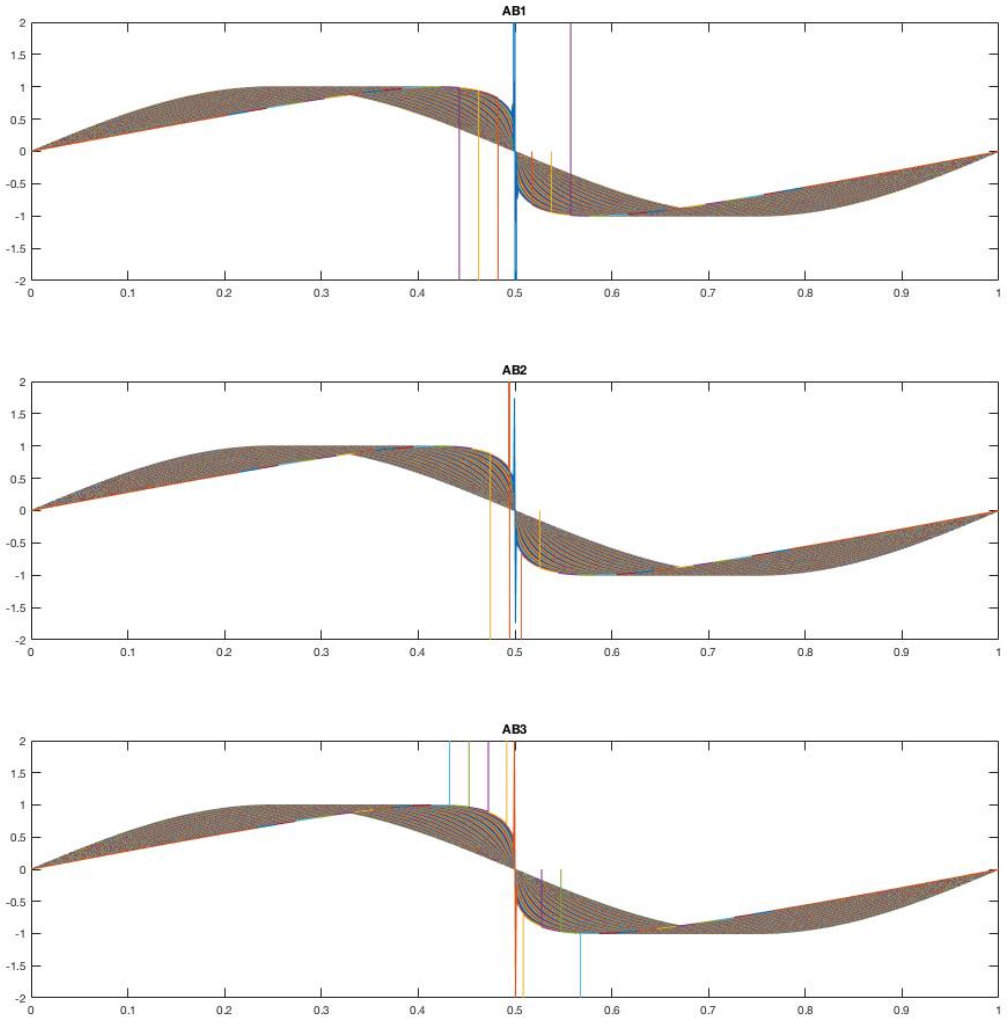

Figure 12: $J = 1000$, $N = 9000$, $dt = 0.0001$. Notes: Allowing this investigation to continue to the bitter end, we see it blowing up in the usual way. Thus, we conclude the crisis that arose in Figure 9 was due to improper temporal resolution.