ENME 631 Numerical Methods

Name: Henry Stoldt
ID: 10127324
Date: Thursday Oct. 24th, 2019
Language: Julia

The file "Assignment 3.jl" runs all of code requested in Q1-3. Explanation and comments below.

Question 1:

The Code:

The code defining question-specific functions required for this question is in "Assignment3Q1.jl". The Explicit Euler, Implicit Euler, and RK4 solvers I coded are general, and are therefore in "ODESolvers.jl". Each solver accepts 4 arguments:

- An array of initial values, one for each variable being integrated
- An array of derivative functions, one for each of the variables. Derivative functions are expected to accept all variables as arguments.
- A marching step size
- A number of marching steps to compute The solvers then return a matrix containing the values of each variable at every marching step.

Explicit Euler:

This solver is very straightforward. At each time step, it evaluates the derivative function for each variable, multiplies the derivative by the time step size, and adds the result of that calculation to each variable's previous value.

Implicit Euler:

This solver was implemented using Newton-Raphson iterations for a precise solution. At each time step, the (i+1) values of each variable are first predicted by a single explicit euler computation, then corrected by one or more iterations of a Newton-Raphson solver. The version of Newton-Raphson used here computed the Jacobian numerically, using a function developed for Assignment #2 (ddx). The residual vector for the Newton-Raphson matrix is obtained by comparing the current guess of the next set of variables with the result of implicit euler using that guessed set of variables. The matrix is the solved by Gauss Elimination, also using a function developed for Assignment #2.

Runge-Kutta 4th order:

This solver is also relatively straightforward, it simply follows the RK4 algorithm, which doesn't leave many implementation choices to be made. k1-k4 are computed and the next time step's values are computed as prescribed by the RK4 method.

Solution:

Both the explicit euler and the rk4 solutions exhibit growing oscillations in all three variables. Somewhat regularly shaped, but irregularly spaced spikes appear in the results after t=15. The solution changes visibly as b is increased by 0.001, indicating that, using these solvers, the solution is very sensitive to the value of b. The implicit euler method does not exhibit any of the above characteristics. Oscillations in each of the variables decrease in magnitude over time and changing b by 0.001 did not appear to have had an outsized impact on the solution. This demonstrates the superior stability of the implicit euler scheme compared to the explicit euler and rk4 schemes.

Question 2:

The Code:

This question reuses the explicit euler solver developed for Q1 and adds a modified euler solver, also in "ODESolvers.jl". The modified euler solver is also general, and accepts the same arguments as those detailed for the solvers in Q1. It works using a two-step method. The predictor applies explicit euler to each variable, after which the second step uses the result of the predictor to estimate the value of the derivative at the next time step. Finally, next value of each variable is incremented by the time step multiplied by the average of the derivatives computed by the first and second prediction, giving a second order method.

Solution:

With a sufficiently small time step, the results of this computation appear to be a very small steady oscillation (can only see this on the plots if you zoom in), with the values of each variable remaining near 1 and 2. As the time step is increased, the explicit euler method develops growing oscillations and diverges much more quickly than the modified euler method, which is more stable due to using predicted values at the next time step in its calculation. Results at four time step sizes are plotted for the first 250 seconds of reaction. With a time step size of 0.01, both methods show quite stable behaviour. At dt=0.04, explicit euler shows a trend of clearly growing oscillations. At dt=0.16, explicit euler produces large oscillations, while modified euler is still stable. Finally, at dt=0.64, explicit euler completely diverges and modified euler starts exhibiting growing oscillations, slightly larger than those exhibited by the explicit euler method at dt=0.04.

Question 3:

Assumptions:

- Pr = 0.71 (http://www.thermopedia.com/content/1053/)

The Code:

Unfortunately I didn't have time to write a generalized solver for this type of problem. The code in "Assignment3Q3.jl" is custom for this particular problem. It uses the equilibrium method so solve both nonlinear ODEs simultaneously. The equations were solved using the following process:

1. Inital values of z and t are guesssed
2. Linearized equations by lagging lowest order terms ("iteration" method, pg. 472 of Hoffman)
3. Lagged coefficients calculated based on last timestep's values
4. Matrix assembled using second order or higher finite difference equations for each of the derivatives. One large matrix was asssembled containing both the zeta and t equations

5. One additional equation is added for each boundary condition. For dirichlet conditions, the equation simply sets the value of a certain node. For Neumann conditions, the derivative was discretized using a second order or higher finite difference, which was then inserted into the additional equation.
6. Matrix is solved using Gauss Elimination function from Assignment 2 (with some row/column reordering this could use a Thomas algorithm type solver instead, but using Gauss here for simplicity)
7. Under-relaxation is applied by averaging the newly computed values with last timestep's values.
8. The process is repeated until the largest change in all values of z and t from one iteration to the next is < 0.00001

Solution:

The solver computes a smooth solution which appears to satisfy all boundary conditions. Unfortunately the solution appears to be somewhat grid-dependent, with the zeta curve especially shifting noticeably as more nodes are used. Unfortunately using a large number of nodes is impractical since the solver currently uses Gauss Elimination. Based on observations between 0 and 800 nodes in the solution, it appears as though the values are converging, but somewhat slowly. Using 800 nodes, the value of dT/dn at eta = 0 was approximately: -0.16. As mentioned previously, this number is not grid independent. One concern I have about the solution is that the text of question 3 mentions that fluid velocity at infinity is zero. I've interpreted this to mean the V=0 at infinity. Since it seems like we're approximating infinity with eta=20 here, that would indicate that zeta needs to = 0 when eta=20, which is not the case in my solution. I'm not currently able to find any issues in my code though and all the specified boundary conditions are satisfied, so perhaps I'm misinterpreting V or the text does not agree with the stated boundary conditions for some reason, or perhaps there's a bug in the code somewhere.

Notes:

The file "LinearSolvers.jl" containing the Gauss Elimination solver is not being resubmitted, it was already submitted in Assignment 2 and has not changed. The file "NonLinearSolvers.jl" containing the "ddx" method for numerically evaluating derivatives (used in Newton-Raphson iterations) is not being resubmitted, it was already submitted in Assignment 2 and has not changed.