CS 322 Project 3: Springies - Backward Euler (Starred Problem)

1 Background

Forward Euler, Midpoint, and the Runge-Kutta integrators have one thing in common - they are all *explicit* methods for advancing an ODE. An explicit method has a formula for $\mathbf{u}_{n+1}$ in terms of only $\mathbf{u}_n$ and $G(\mathbf{u}_n, t)$. For instance, Forward Euler when applied to the ODE $\dot{\mathbf{u}} = \mathbf{G}(\mathbf{u}, t)$ gives

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{G}(\mathbf{u}_n, t)$$

Here we have straightforward equations for the next state given our current state. Note in particular that we are using the function evaluated at the current state to update the new state. Suppose instead we revise these equations:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{G}(\mathbf{u}_{n+1}, t_{n+1})$$

Now, we are using the ODE function evaluated at the *new* state in order to determine what the new state should be. This gives us what is commonly known as the Backward Euler scheme for integration. Note that because our equation for the new state uses the new state itself, we have an *implicit* method instead of an explicit method. This means that we no longer have a simple formula for the new state given the current state.

Let $\Delta\mathbf{u} = \mathbf{u}_{n+1} - \mathbf{u}_n$. Then we can rewrite the above equation as

$$\Delta\mathbf{u} = h\mathbf{G}(\mathbf{u}_n + \Delta\mathbf{u}, t_{n+1})$$

One approach to solving this is to take the first order Taylor expansion of the right hand side about $\mathbf{G}(\mathbf{u}_n, t_n)$:

$$\Delta\mathbf{u} = h(\mathbf{G}(\mathbf{u}_n, t_n) + \frac{\delta\mathbf{G}}{\delta\mathbf{u}}\Delta\mathbf{u} + \frac{\delta\mathbf{G}}{\delta t}\Delta t)$$

where $\frac{\delta\mathbf{G}}{\delta\mathbf{u}}$ is the Jacobian matrix of derivatives of each component of $\mathbf{G}$ with respect to each component of $\mathbf{u}$. We note that in this assignment, $\mathbf{G}$ does not depend on t, and so the derivative with respect to t will be the zero vector. Doing some algebraic manipulations gives us:

$$\Delta \mathbf{u} = h\mathbf{G}(\mathbf{u}_n, t_n) + h\frac{\delta \mathbf{G}}{\delta \mathbf{u}}\Delta \mathbf{u}$$
$$\Delta \mathbf{u} - h\frac{\delta \mathbf{G}}{\delta \mathbf{u}}\Delta \mathbf{u} = h\mathbf{G}(\mathbf{u}_n, t_n)$$
$$\left(\mathbf{I} - h\frac{\delta \mathbf{G}}{\delta \mathbf{u}}\right)\Delta \mathbf{u} = h\mathbf{G}(\mathbf{u}_n, t_n)$$

Note that this is a system of linear equations – we have a matrix on the left hand side and a vector on the right hand side, and we are solving for the change in the state vector. You should also note that for this assignment the matrix is $4n \times 4n$ and the change in state vector is $4n$, where n is the number of particles, because we are working in 2D and there are 2 position components and 2 velocity components in the state for each particle . Once we set up and solve this system for $\Delta \mathbf{u}$, we can get the value of $\mathbf{u}_{n+1}$ in a straightforward manner.

One catch (and the reason why explicit schemes are used whenever possible) is that the derivatives of each component of the the ODE function $\mathbf{G}$ with respect to each component of the state must be computed. However, this matrix will in general have a specific structure in this assignment. For instance, the derivative of the components of $\mathbf{G}$ corresponding to some particle i with respect to the components in the state corresponding to some particle j will be nonzero if and only if there is some force acting between them — in the case of the assignment, if there is a spring between the two particles. Thus, you can use the sparse matrix libraries built into MATLAB to build the right hand side matrix (such as `sparse` and `speye`, etc). In general, this may or may not result in a faster linear system to solve — if you have the time, you may want to experiment and see whether sparse or dense matrices result in faster execution.

In order to implement Backwards Euler, you will have to

1. Change your ODE function to optionally return the sparse matrices corresponding to the derivatives of each component of $\mathbf{G}$ with respect to $\mathbf{u}$ (they should only be calculated if needed)

2. Implement the code to solve the above linear system given the derivative matrix returned by your ODE function