AE588 Assignment 5 - akshatdy

5.1.a

- Do you get any complex-step derivatives with zero error compared to the analytic reference?
  - Yes I do, and those are artificially replaced with 1e-16 to get a nicer looking plot that is continuous
- What is the smallest step you can use before you get underflow?
  - The smallest step I can use is 1e-323, after that I get a "invalid value encountered in scalar divide" when the functions try to divide by the step size, and for step sizes smaller than that, the output is nan
  - The point at which different algorithms are affected by underflow are:
    * Forward difference: 1e-15
    * Central difference: 1e-15
    * Complex step: It doesnt underflow in the sense that the derivative doesnt go to 0, but it starts deviating from the actual gradient at 1e-308
- What does that mean, and how should you show those points on the plot?
  - For finite difference, this means that the subtractive cancellation makes the difference = 0 which causes the overall gradient to reduce to 0
  - For central difference, this means that the division encounters a 0 and gets a nan as a result of the computation
  - When plotting the errors of the derivatives
    * Top end: Plotting in log ensures that large values generated by very small step sizes for forward and central finite difference do not obscure our view of the smallers errors generated by larger step sizes
    * Bottom end: Reassigning all the 0 errors of complex step to a value of 1e-16 makes the graph continuous and easier to read as well
- Estimate the value of h required to eliminate truncation error in derivative using the equations in the textbook.
  - From the textbook, the optimal step size for the forward finite difference is approximately $\sqrt{\epsilon_f}$, where ϵ_f is the precision of f, and for the central difference, the optimal step size scales approximately with $\sqrt[3]{\epsilon_f}$
  - Precision of f can be found with the `math.ulp()` function, so the optimal h is:
    * Forward difference: 2.9802322387695312e-08
    * Central difference: 9.61243476787471e-06

- Is this estimate consistent with your plot?
  - Yes this estimate is consistent with the plot

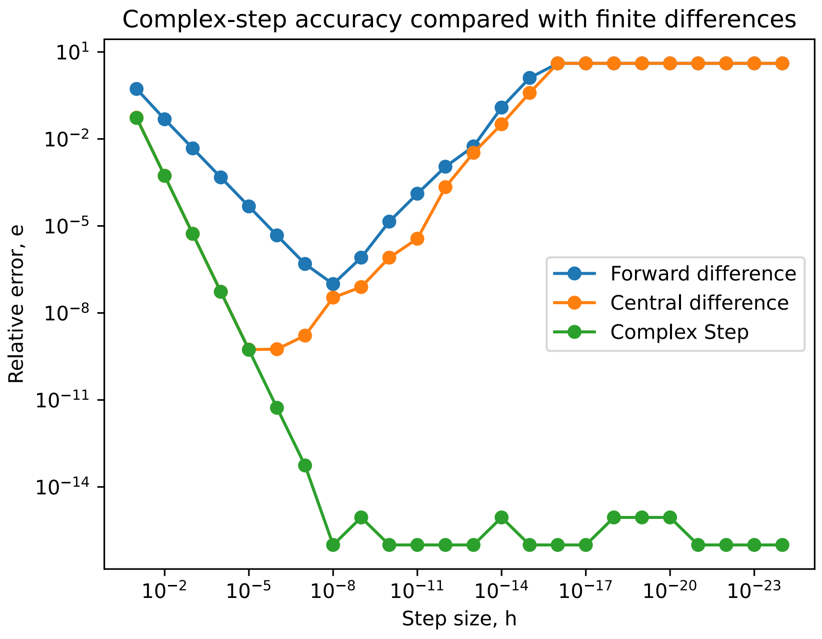

Figure 1: Comparison Graph

Full plot with underflow:

5.1.b

The AD implementation consists of new functions for all the operators required in the given problem. All these functions take in a new data type, that consists of the value and a derivative. The derivative is initialized to 1 as the seed so that subsequent operations with the data can utilize the chain rule effectively. The problem equation is then re-written using the the AD operators and we get the value and derivative when we evaluate the problem function directly.

Using AD, at x = 1.5, fx = 4.497780053946162, dx = 4.053427893898621

The derivative matches what I get from the complex step method, the difference is 0

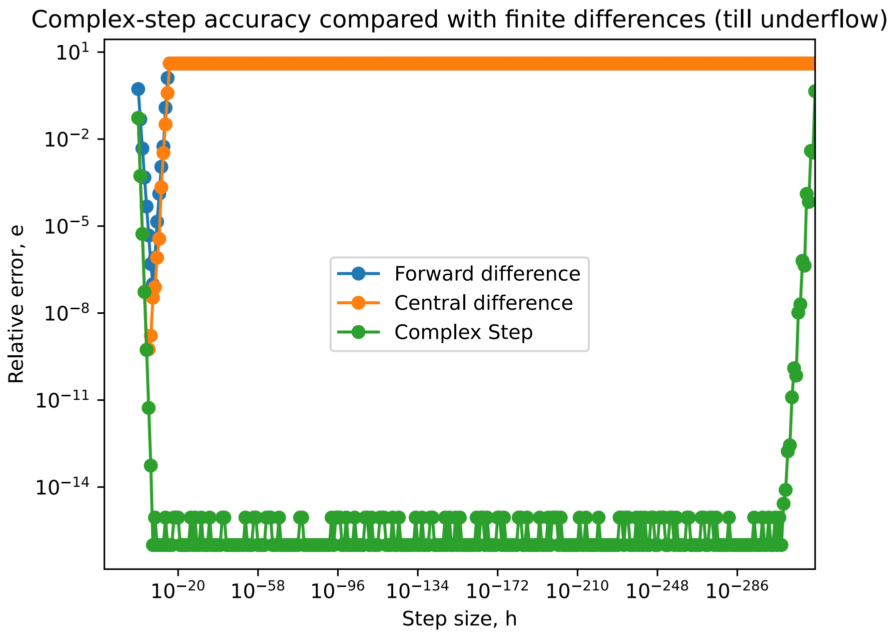

Figure 2: Comparison Graph

5.2

$$\begin{aligned} x_1 &= M \\ u_1 &= E \\ r_1(x_1) &= M \\ r_2(u_1) &= E - esin(E) - M = 0 \\ r_3(x_1, u_1) &= f - E + M \end{aligned}$$

The forward system of the UDE is:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 - ecos(E) & 0 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \frac{du_1}{dx_1} & \frac{du_1}{dr_2} & 0 \\ \frac{df}{dx_1} & \frac{df}{dr_2} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Solving the first column to get $\frac{df}{dx_1}$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 - ecos(E) & 0 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \frac{du_1}{dx_1} \\ \frac{df}{dx_1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

We have

$$\begin{aligned} 1 &= 1 \\ -1 + \frac{du_1}{dx_1}(1 - ecos(E)) &= 0 \\ 1 - \frac{du_1}{dx_1} + \frac{df}{dx_1} &= 0 \end{aligned}$$

Solving for $\frac{df}{dx_1}$, we get

$$\frac{df}{dx_1} = \frac{1}{1 - ecos(E)} - 1$$

so

$$\frac{df}{dM} = \frac{1}{1 - ecos(E)} - 1$$

Verification with finite differences match the analytical results

- at M=0.5, df/dM is:
  - UDE: 0.4202054768667982
  - FD: 0.4202054315616266
- at M=1, df/dM is:

  - UDE: -0.07958665799511588
  - FD: -0.07958664394180914
- at M=1.5, df/dM is:
  - UDE: -0.26214792220067595
  - FD: -0.2621479411324401
- at M=2, df/dM is:
  - UDE: -0.349858262004625
  - FD: -0.3498582623606694
- at M=3, df/dM is:
  - UDE: -0.41092304468673113
  - FD: -0.4109230378190887

5.3

Using equation 6.42 for Example 6.12

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \frac{\partial f}{\partial u}\frac{\partial r}{\partial u}^{-1}\frac{\partial r}{\partial x}$$

$$\begin{aligned} &where \\ &\frac{\partial f}{\partial x} = 0 \\ &\frac{\partial f}{\partial u} = S \\ &\frac{\partial r}{\partial u} = K \\ &\frac{\partial r}{\partial x} = using\ FD/CS \end{aligned}$$

$$\begin{aligned} For\ Direct\ method,\ we\ need\ \phi\ such\ that \\ K\phi &= \frac{\partial}{\partial x}Ku \\ which\ can\ be\ obtaied\ using\ a\ linear\ solver \\ We\ can\ then\ use\ \phi\ to\ get \\ \frac{df}{dx} &= -S\phi \end{aligned}$$

$$\begin{aligned} For\ Adjoint\ method,\ we\ need\ \psi\ such\ that \\ K\psi &= S^T \\ which\ can\ be\ obtaied\ using\ a\ linear\ solver \\ We\ can\ then\ use\ \psi\ to\ get \\ \frac{df}{dx} &= -\psi^T\frac{\partial}{\partial x}Ku \end{aligned}$$

Unlike the example 6.12, we actually did not need to go column by column or row by row to compute ϕ or ψ since they can be solved in a single call with the numpy linear solver. This is why I skiped the i and j subscripts in the equations.

Relative errors of derivatives

Columns are the algorithms being compared, rows are the algorithms being compared to

Mean Error Cross section area = 0.00051

	FD	CS	DT
FD			
CS	1.5693885405866314e-05		
DT	5.267695884326408e-09	1.5694225171944888e-05	
AJ	5.267695954118095e-09	1.56942251720772e-05	2.0698942265641576e-15

Cross section area = 0.0051

	FD	CS	DT
FD			
CS	1.5740799499070659e-06		
DT	2.896905846095844e-08	1.5765855570527423e-06	
AJ	2.896905875393636e-08	1.5765855567551157e-06	1.560493808446306e-15

Max Error Cross section area = 0.00051

	FD	CS	DT
FD			
CS	1.76301404257226e-05		
DT	7.345428070377964e-08	1.759104891132644e-05	
AJ	7.345428458456883e-08	1.759104891275623e-05	3.5909090956818885e-14

Cross section area = 0.0051

	FD	CS	DT
FD			
CS	2.0315278996614934e-06		
DT	3.354622427716693e-07	1.7679381964847262e-06	

	FD	CS	DT
AJ	3.354622481970304e-07	1.767938198420282e-06	1.9663696351694963e-14

Discussion As expected, DT and AJ perform similarly, while FD and CS perform worse than those methods. It is interesting that DJ and AJ are not exactly the same, because mathematically they should be. This could be due to the loss of precision adding up as we multiply different matrices together for DT and AJ. The more mathematical operations we do, the more accuracy we will lose. Also, both of these methods are using a linear solver to solve a different set of equations, where the precision loss might also build up.

Another interesting observation is that according to the theory, the CS method should be very accurate, but it is not as close to the DT and AJ methods as expected.

Computational cost of derivatives

These tests are averaged over 100 function calls

FD = 4.062857627868652 ms

CS = 4.0137457847595215 ms

DT = 5.500221252441406 ms

AJ = 5.625064373016357 ms

All the algorithms take around the same amount of time, which makes sense given that they are all making 2 calls to the function for the finite difference steps (2x effort per call for complex step for the imaginary part). Ideally the finite difference would not be needed if we could calculate the partial derivatives symbolically and that would save a lot of computation time and make DT and AJ more efficient than the rest. In addition to the function calls, DT and AJ also involve using a linear solver, which adds to their run time, making them slighly slower.