

Devito Finite Differences - Important Quirks

Akhil Krishna Mohan

June 28, 2022

Introduction

Devito is a powerful Domain-Specific Language that allows us to solve PDEs using finite difference stencils. It handles generating forward and adjoint propagators in C for efficient time-stepping. In particular, if we define a function u of space and time in our domain, $u.dx$ defines the first derivative of u in the x -direction, and $u.dy$ similarly in the y -direction. Problems arise when we need accurate stencils of higher orders. Typically, for a second derivative in the x -direction, we would use

$$u.dx2 \equiv \frac{\partial^2}{\partial x} u_{(x,y)} \approx \frac{u_{(x+1,y)} + u_{(x-1,y)} - 2u_{(x,y)}}{\Delta x^2} \quad (1)$$

This is the stencil generated when $u.dx2$ is called, and this is great. However, we get different behaviour if we use the notation $(u.dx).dx$, which instead approximates the above as

$$(u.dx).dx \equiv \frac{\partial^2}{\partial x} u_{(x,y)} \approx \frac{u_{(x+2,y)} + u_{(x,y)} - 2u_{(x+1,y)}}{\Delta x^2} \quad (2)$$

This discrepancy is slight, and for the most part has workarounds, but dealing with this issue has its nuances. We will build up to the behaviour of `grad` and `div`, but we choose to examine only x -derivatives (without loss of generality) until that point, for brevity.

Devito FD Mechanisms

The discrepancies in (1) and (2) are to be expected, since by default $u.dx$ is equivalent to $u.dxr$, or a forward difference. This means that

$$u.dx \equiv u.dxr \equiv \frac{\partial}{\partial x} u_{(x,y)} \approx \frac{u_{(x+1,y)} - u_{(x,y)}}{\Delta x} \quad (3)$$

Accordingly, we have

$$u.dxl \equiv \frac{\partial}{\partial x} u_{(x,y)} \approx \frac{u_{(x,y)} - u_{(x-1,y)}}{\Delta x} \quad (4)$$

and

$$u.dxc \equiv \frac{\partial}{\partial x} u_{(x,y)} \approx \frac{u_{(x+1,y)} - u_{(x-1,y)}}{2\Delta x} \quad (5)$$

All of these are defined exactly this way when we set the spatial order of u to be 2 in Devito. Out of the above, only (5) seems to be a "true" second-order accurate scheme. However, one can argue that (3) and (4) are also second-order accurate. They are simply defined at $u_{(x+\frac{\Delta x}{2},y)}$ and $u_{(x-\frac{\Delta x}{2},y)}$ (ghost nodes on the grid). On the surface, this seems like a good workaround, but we have issues when we combine functions that are defined at different places. We consider the acoustic isotropic wave equation in 2D, but propagating only in x . The maths is identical for each dimension, but we look only at x -derivatives.

$$\frac{1}{v^2} \frac{\partial^2 u}{\partial t^2} - \frac{1}{b} \frac{\partial}{\partial x} \left(b \frac{\partial u}{\partial x} \right) - \frac{1}{b} \frac{\partial}{\partial y} \left(b \frac{\partial u}{\partial y} \right) = s \quad (6)$$

with s as a source term and b as buoyancy. Looking at the second term, we would aim to represent it with a stencil that is a function of $u_{(x-1,y)}$, $u_{(x,y)}$, and $u_{(x+1,y)}$, while also using $b_{(x-1,y)}$, $b_{(x,y)}$, and $b_{(x+1,y)}$.

We examine different ways of specifying the red term in (6) in Devito -

$$1. \quad \text{pde} = m * u.\text{dt2} - 1/b * (b * u.\text{dx}).\text{dx}$$

gives us a stencil that is a function of $u_{(x,y)}$, $u_{(x+1,y)}$, and $u_{(x+2,y)}$, using $b_{(x,y)}$ and $b_{(x+1,y)}$. This is asymmetric (i.e. no $(x-1, y)$ terms), but also highlights a "bad" multiplication - b is defined at grid point (x, y) but $u.\text{dx}$ is shifted, i.e. defined at $(x + \frac{\Delta x}{2}, y)$.

2. We can set out to fix this by replacing $u.\text{dx}$ with $u.\text{dxc}$. This will mean that b , defined "at" the grid points, will be multiplied by $u.\text{dxc}$, also defined at the grid points.

$$\text{pde} = m * u.\text{dt2} - 1/b * (b * u.\text{dxc}).\text{dxc}$$

This stencil solves the problem of "bad" multiplications, but expands the stencil to be a function of $u_{(x-2,y)}$, $u_{(x,y)}$, and $u_{(x+2,y)}$, while using $b_{(x-1,y)}$, $b_{(x,y)}$, $b_{(x+1,y)}$. This is correctly defined, however, we can achieve second-order accuracy for this operation without using $u_{(x-2,y)}$ and $u_{(x+2,y)}$. In this stencil, we have an optimal stencil size, but we actually have 2 disjoint ("checkerboarded") grids (in 1D), since $u_{(x,y)}$ does not depend on $u_{(x+1,y)}$ and $u_{(x-1,y)}$.

3. Another approach would be to combine .dxr with .dxl . We could define

$$\text{pde} = m * u.\text{dt2} - 1/b * (b * u.\text{dxr}).\text{dxl}$$

This has both of the previously mentioned drawbacks in that it uses the larger stencil, including $u_{(x-2,y)}$, $u_{(x-1,y)}$, $u_{(x,y)}$, $u_{(x+1,y)}$, and $u_{(x+2,y)}$, but uses shifted values of b , i.e. $b_{(x-2,y)}$, $b_{(x-1,y)}$, $b_{(x,y)}$. This stencil is then asymmetric in b and also seems to use the wrong kinds of multiplication.

4. We can also expand the derivatives in b algebraically before specifying the formulation in the PDE. This would look like

$$\text{pde} = m * u.\text{dt2} - 1/b * (b * u.\text{dx2} + b.\text{dxc} * u.\text{dxc})$$

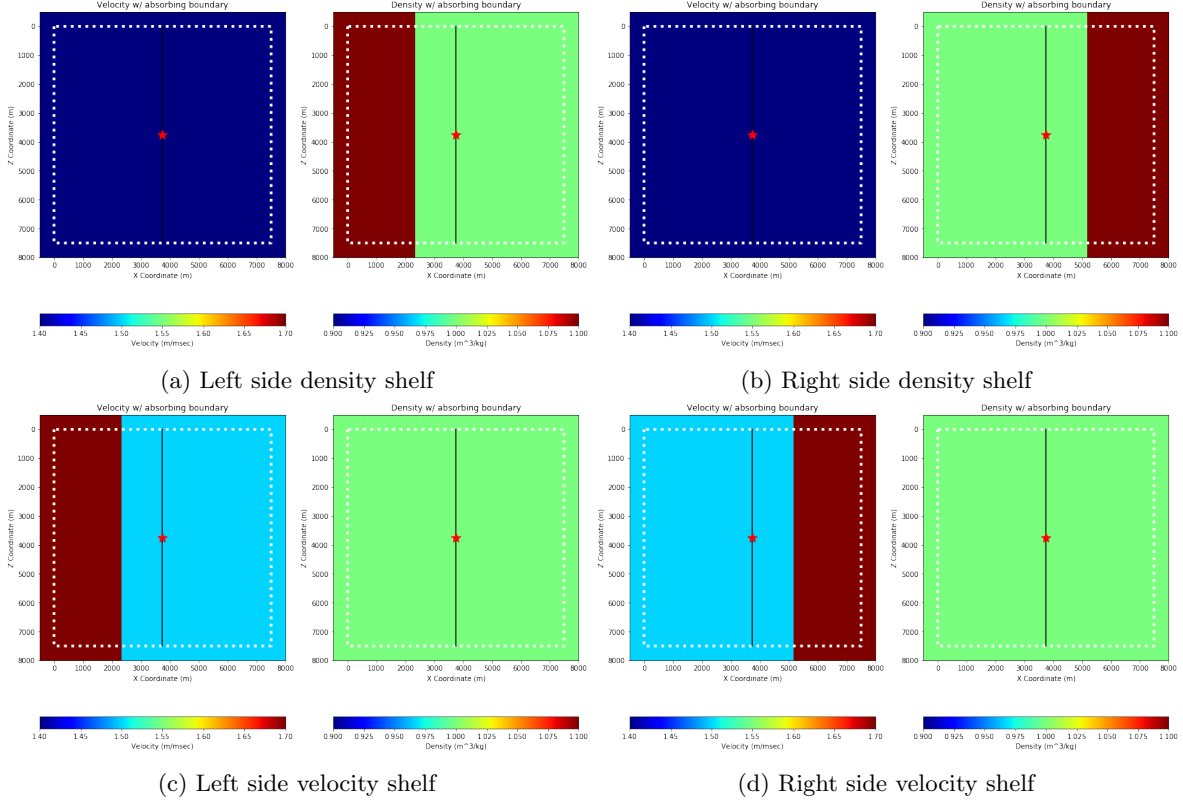
This gives us a stencil using $u_{(x-1,y)}$, $u_{(x,y)}$, $u_{(x+1,y)}$, $b_{(x-1,y)}$, $b_{(x,y)}$, and $b_{(x+1,y)}$, which we are after.

These FD models are examined in more detail in [this notebook](#) and results have been collated in [this file](#).

Why is this a problem?

Apart from "bad" multiplications and larger-than-necessary stencils, there are more concerning problems with Devito.

- The checkerboarding pattern described above can be exploited to do asynchronous grid-updates and save memory, but by default Devito does not do this. For time order k , Devito keeps $k+1$ copies of the function and updates them completely. Thus, we are unable to exploit this for performance, and also end up with essentially 2^n disjoint grids within our computational grid, where n is the number of dimensions we formulate in.
- Consider the code [example](#) provided in Devito's examples package. It defines a self-adjoint operator. However, as per a testing [notebook](#) I created, I was able to analyze if the propagation was symmetric in both directions.



I found that when the velocity field is asymmetric, we see no difference in wave propagation, but when the buoyancy is asymmetric, the resulting waveform is shifted by a small amount, i.e. the propagator had a bias to either the left or right side of the domain. This happens because of the above described FD models, and the in-built Devito implementation of `grad` and `div`. We never take a derivative of the velocity in the example, therefore the behaviour is unbiased. We do take derivatives of the buoyancy, and this leads to a biased propagation. The fact that propagation is not the same in the $+x$ and $-x$ directions should be critical enough to warrant our attention.

- Since the implementations of `grad` and `div` inherently rely on the above Finite Difference(FD) models, they will suffer from similar shifting problems, stopping us from specifying PDEs in a convenient format using Devito's inbuilt operators. This means we need to be careful with how we specify our equations. Being unable to use Devito's inbuilt `grad` and `div` functions adds much clutter to the PDE formulation, and we would likely need to look into automatic code generation to make sure schemes are defined correctly.