Numerical details for implementing a 1D advection in solver in OpenFD using SBP high order finite differences

March 31, 2018

In this document we describe some numerical details that are useful for understanding how the example code works, and for asserting that the implementation is correct.

1 Continuous problem

Consider the 1D advection

$$\begin{aligned} u_t + au_x &= 0, 0 \leq x \leq 1, \\ u(x,0) &= f(x), \\ u(0,t) &= g(t). \end{aligned} \tag{1}$$

If we restrict attention to $a > 0$ and let $g(t) = f(-at)$ than this PDE describes a solution $u(x,t)$ that is given by propagating the initial condition $f(x)$ towards the right at the speed a. That is, the solution is given by $u(x,t) = f(x-at)$.

2 Numerical scheme

In the script *kernel.py*, we use the kernel generator capabilities of OpenFD to generate GPU code for the following semi-discrete approximation of the problem (1):

$$\frac{d\mathbf{u}}{dt} + aD\mathbf{u} = -\tau \mathbf{e}_0(u_0 - g(t)). \tag{2}$$

In this approximation, $\mathbf{u}^T = [u_0, u_1, \ldots u_{n-1}]$ is a *gridfunction* that defines the numerical solution at each grid point. Furthermore, $D = H^{-1}Q$ is a *summation-by-parts* (SBP) finite difference operator that discretizes the first derivative. The boundary condition $u(0,t) = g(t)$ is implemented by using the *Simultaneous-Approximation-Term* (SAT) method. This method weakly enforces the boundary condition, and is accomplished by the term in the right-hand side of (2). The SAT term consists of a scaling parameter τ that is determined for stability, a restriction operator $\mathbf{e}_0$, and the boundary condition $u_0 - g(t)$. The scaling parameter is set to $\tau = H_{00}^{-1}$, which is the reciprocal of the first coefficient in

the diagonal norm matrix H. The role of the restriction operator $\mathbf{e}_0$ is to ensure that the SAT term only acts on the boundary $x = 0$. That is, $\mathbf{e}_0^T \mathbf{u} = u_0$.

We also need to perform a discretization in time to complete our numerical scheme. For this purpose, we have chosen to use a low-storage Runge-Kutta scheme. This method can roughly be summarized as follows for the advancement from time t to time $t + \Delta t$. For each stage $s = 1, 2, \ldots, n$, do

$$\mathbf{du} := a_s \mathbf{du} + F(\mathbf{u}, t + c_s \Delta t) \tag{3}$$

$$\mathbf{u} := \mathbf{u} + \Delta t b_s \mathbf{du} \tag{4}$$

In this recipe, **du** is extra, temporary storage that we refer to as the *rates* of the solution **u**. The coefficients a_s, b_s, c_s are Runge-Kutta coefficients that are known ahead of time. The function $F(\mathbf{u}, t)$ is the spatial discretization. That is,

$$F(\mathbf{u}, t) = -aD\mathbf{u} - \tau \mathbf{e}_0 (u_0 - g(t)). \tag{5}$$

3 Kernel generation

To prepare this numerical description for kernel generation using OpenFD, we split (2) into two parts: (i) computation of the PDE without SAT term, and (ii) computation of the SAT term. Whenever we want to generate a kernel, there are essentially three aspects we need to be aware of. These are

- Input arguments
- Output arguments
- Bounds

For the spatial part of the PDE computation, there is only one input argument which is the spatial discretization of this PDE. There are as many output arguments as input arguments, and the only output argument is the variable that we wish to assign to.

The bounds determine the grid points that we wish to compute for, and depend on what type of computation to perform. What influences the specification of the bounds is the mixture of matrix vector computation and convolution computation present in the derivative operator D. For points near the boundary, this operator applies one-sided stencils that changes from point to point, whereas in the interior, the same central stencil is repeated at each interior point. To handle this computational heterogeneity, OpenFD splits the computation into three separate parts, called regions. The typical scenario is to have a left and right boundary region, and a large interior region. In order to perform this splitting into regions, the kernel generator in OpenFD needs to known how many points that each region uses. The number of boundary points depend on which operator is used. For example, a traditional, second order, SBP finite difference operator would only use one boundary point per boundary. Consistent bounds for this operator, on a grid with n points, would be defined by passing the following to the kernel generator

```
bounds = Bounds(size=n, left=1, right=1).
```

4 Convergence test

To test that our implementation is correct, we use the *method of manufactured solutions* (MMS). This method relies on the construction of an a priori known solution that is chosen to be smooth, and a modification of the governing equations and boundary conditions such that this solution is satisfied.

In this case, we have chosen the manufactured solution to be

$$u^*(x,t) = \sin(kx - \omega t), \tag{6}$$

where $k = \omega/a$ is the wave number. This solution satisfies the problem (1) without any modifications to the PDE. In the solver script *solver.py*, we initialize the numerical solution $\mathbf{u}$ by setting it to $\mathbf{u} = u^*(\mathbf{x}, 0)$ and we set the time dependent boundary data, $g(t)$, to $g(t) = u^*(0,t)$.

Note that we use the supercript $*$ to distinguish the exact, manufactured solution from the numerical solution.

In the script, *convergence.py*, (we have chosen to define the error e in the numerical solution as the difference between the numerical $\mathbf{u}$ and exact $u^*(\mathbf{x},t)$ solution, and we have chosen to measure this error in the weighted, discrete l2-norm

$$\|e\|_h = \|u - u^*\|_h = \sqrt{e^T H e}. \tag{7}$$

In this definition, the norm matrix H is diagonal and is associated with the SBP operator used to discretize in space. For a traditional, second order, SBP finite difference operator, the norm of the error is

$$\|e\|_h = \sqrt{\frac{h}{2}e_0^2 + h\sum_{i=1}^{n-2} e_i^2 + \frac{h}{2}e_{n-1}^2},$$

where h is the grid spacing. Using this definition of norm of the error, we compute the convergence rate q by subsequently refining the grid by halving the grid spacing:

$$q_j = \log_2(\|e\|_j / \|e\|_{j+1}).$$

In the definition of the convergence rate q_j, $\|e\|_j$ is the norm of the error on a grid j with grid spacing $h_j = h(1/2)^j$.