P-ADAPTIVE METHODS FOR HYPERBOLIC PDES

DEVIN LIGHT AND SCOTT MOE

1. Introduction

In this work we investigate using the Discontinuous Galerkin method with p-adaptivity to give us variable resolution. The Discontinuous Galerkin Finite Element Method (DGFEM) is a high order extension of the Godunov Method. Instead of increasing grid resolution in the typical way by locally dividing cells, we could add more degrees of freedom to a cell and define a higher order scheme on that cell. This is known as p-adaptivity (for polynomial adaptivity) because it typically involves using basis functions of different polynomial orders on different cells. Traditional mesh adaption strategies are then known as h-adaptivity.

We will introduce the Discontinuous Galerkin method and explain its analogies to the Finite Volume method. Following that we will examine the requirements for effective dynamic p-adaptivity. This is somewhat of an open challenge in the literature although there are several papers that have explored this topic. We will attempt to mimic how the CLAWPACK software package implements Adaptive Mesh Refinement (AMR) for hyperbolic PDEs. An exploration of the performance of this strategy will be preformed on some simple problems. This results in a simple p-adaptive strategy based on surrounding any high-order regions of a domain with buffer cells. This strategy is surprisingly effective even when not implemented with any sort of adaptive time-stepping.

2. Discontinuous Galerkin Method

The Discontinuous Galerkin method is a arbitrary order extension of the first order Finite Volume method. This method is actually a hybrid method using the technologies of both the Finite Element and Finite Volume families of methods. In a Finite Element method we typically use our PDE to construct a variational form that can be enforced on some finite dimensional function space. This is done by multiplying by a test function ϕ, and integrating by parts.

Let us define a closed domain to be Ω and our test function to be ϕ Also let us assume that we are solving a hyperbolic PDE that can be written using the differential form in equation (1).

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})}{\partial x} = 0 \tag{1}$$

We can multiply equation 1 by a test function and integrate over some domain Ω. We will then integrate by parts to obtain the weak form in equation (2).

$$\begin{aligned} &\int_\Omega (\phi(\mathbf{x})\frac{\partial q}{\partial t} + \phi(\mathbf{x})\mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})_x)d\Omega = \\ &\int_\Omega \phi(\mathbf{x})\frac{\partial q}{\partial t}d\Omega + \phi(\mathbf{x})\mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})|_{\partial\Omega} - \int_\Omega \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})\phi_x d\Omega = 0 \end{aligned} \tag{2}$$

where $\mathbf{q}$ is our solution and ϕ is a test function. In a typical Finite Element method Ω would be the entire domain. However for hyperbolic PDEs this usually does not work well. A technical explanation of why is beyond the scope of this document, the important thing is that to solve these hyperbolic PDEs we will borrow ideas from the Finite Volume method. We will use a function space defined locally on cells in the domain, and solve a separate version of equation (2) on each cell.

For this simple 1-d problem we will split up our domain into subcells $\Omega_k = (x_{k-1/2}, x_{k+1/2})$. This requires solving equation (3) on each cell.

$$\int_{\Omega_k} \phi(\mathbf{x}) \frac{\partial q}{\partial t} d\Omega_k = -\phi(\mathbf{x}) \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})|_{\partial\Omega_k} + \int_{\Omega_k} \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t}) \phi_x d\Omega_k \tag{3}$$

This equation should look similar to the equations used to derive the first order Godunov method. The main difference is the term $\int_{\Omega_k} \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t}) \phi_x d\Omega_k$ which comes from the fact that we multiplied by a test function. If ϕ is a constant this reduces to the Godunov method. This method is typically implemented using Runge-Kutta schemes to integrate coefficients in time [2].

2.1. **Choosing a Function Space.** The formulation of the DGFEM presented in equation (3) completely decouples neighboring cells from one another. This removes the typical Finite Element method requirement that our basis functions are C_0. As such we will no longer use piecewise defined Lagrange polynomial functions which are nonzero on multiple cells as is typical in Finite Elements. Instead it is normal to pick our basis functions so that on each cell Ω_k we have a hierarchical orthogonal basis. This is a natural extension of the low order Godunov method. Essentially it means that, instead of just evolving the zeroth order moments (cell averages) of a solution, we are also evolving higher order moments. This also has the nice side-effect of making the typical Finite Element mass matrix diagonal. The mass matrix is the name for the array which results from terms of the form:

$$\int_{\Omega_k} \phi(\mathbf{x}) \frac{\partial q}{\partial t} d\Omega$$

If we use a Legendre polynomial basi and map our interval from Ω_k to $[-1, 1]$ each mode will evolve according to the equation 4.

$$\dot{Q}_i^{(k)} = \frac{2k+1}{\Delta x} \int_{-1}^{1} f\left(q^h\right) \varphi_\xi^{(k)} \, d\xi - \frac{2k+1}{\Delta x} \left[\varphi^{(k)}(1) \, F_{i+\frac{1}{2}} - \varphi^{(k)}(-1) \, F_{i-\frac{1}{2}} \right] \tag{4}$$

2.2. **Evaluating the Cell Boundary Terms.** The DGFEM has the same issue as the Finite Volume method in that it requires the evaluation of the flux function on the boundary between cells where the solution is not defined. We will use the same Riemann problem approach to define $\mathcal{F}(q_l, q_r)$. This works well in practice but notice that it requires somewhat more of a leap of faith because the actual in-cell discretization looks nothing like a Riemann problem. To avoid any complications in our evaluation of the performance of the method we will stick to running things with a simple Lax-Friedrichs flux. However we can simply modify things to use fluctuations so we could use any of the Riemann solvers used in CLAWPACK.

2.3. **Time-stepping.** In this work our focus is not on accuracy of time-stepping schemes. We will just pick one standard time-stepping scheme and ignore the fact that its accuracy does not match our spatial accuracy in most situations. Additionally we will ignore adaptive time-stepping as beyond the scope of this

paper. We will use the well known 3rd order Runge-Kutta scheme. For the ode $y^{'} = f(t, y)$ this looks like

$$\begin{aligned} k_1 =& hf(t, y^n) \\ k_2 =& hf(t + \frac{h}{2}, y^n + \frac{1}{2}k_1 \\ k_3 =& hf(t + h, y^n - k1 + 2k_2)y^{n+1} = y^n + \frac{1}{6}(k_1 + 4k_2 + k_3) \end{aligned}$$

3. P-Adaptivity for polynomial approximation

Consider the situation where we are approximating some function $f(x)$ on an interval $\Omega = (a, b)$ and we have split our domain into a set of N sub-intervals. Say we just wish to adaptively approximate f on each one of these sub-intervals. In this case one natural approach may be to try and compute the orthogonal series expansion of our function.

Let us define $P_i(x)$ as the ith normalized Legendre polynomial. If we assume that our $f \in L_2(\Omega)$ then the local Legendre series of our solution is given by

$$f(x)|_{x \in I_j} = \sum_{i=1}^{\infty} c_i P_i(2\frac{x - x_j}{\Delta x_j})$$

We know from approximation theory that these coefficients should decay at some rate depending on the local regularity of f. If we want to maintain a certain accuracy for our specific cell j then we should expect that we will not need coefficients much smaller than that tolerance. Clearly the number of important coefficients on each cell could vary highly depending on the form of the function.

The idea of p-adaptive approximation is to allow this order to change naturally from cell to cell. This can be done in a very efficient recursive procedure, stopping when coefficients have reached small enough values. We will describe how this works in Algorithm 1 below. First one should define a maximum order that we are willing to use in our approximation, call it N_m. We will then attempt to construct only the important terms in the Legendre series on each cell. We do this by computing Legendre series terms on each cell until consecutive coefficients are below the tolerance, or we reach the maximum order of the series that we are willing to keep. We have borrowed this refinement criteria from the Chebfun software package [4]. In the literature currently existing on p-adaptivity[5, 3] the authors use very similar refinement checks to determine if they have resolved enough coefficients.

4. P-Adaptivity for a Discontinuous Galerkin scheme

After using Algorithm 1 to compute an initial condition we need to actually solve the PDE. A hyperbolic PDE transfers information at a finite speed throughout the domain. This information may change the required order of a cell. To avoid losing information we must allow solutions to use high enough order in cells surrounding cells where the initial condition projection requires high order.

In the literature there have been several approaches to this. In [5] the authors adaptively change the order every time-step in order avoid losing accuracy. However this is feasible for them as they are using semi-Lagrangian time stepping and do not need to take very many time-steps. We would like to do something to allow us

Algorithm 1 Algorithm for p-adaptive approximation

for $I_j \in I = (a, b)$ **do**
  $c^n = \int_{I_j} f(\xi) P_0(\xi) d\xi$
  $c^o = c^n$
  $k = 0$ and $c_k^j = c^n$
  while $\max(|c^o|, |c^n|) > TOL$ and $k < N_m - 1$ **do**
    $c^o = c^n$
    $k = k + 1$
    $c^n = \int_{I_j} f(\xi) P_k(\xi) d\xi$
    $c_k^j = c^n$
  end while
end for

to only reassign polynomial orders after a series of time-steps. In [3] the authors use the rule that neighboring cells cannot have more than a one-degree difference in maximum polynomial order. This may force us to have too many high-order cells in some regions because if we have one cell requiring very high order this would require a wide stencil of high degree polynomials.

We have decided to use a different approach, one modeled after what is done in the CLAWPACK software package when implementing AMR. We do not want to change polynomial degrees every time-step. In order to preserve the accuracy of the solution, then, we will require a finite region around a cell j where the maximum order is at least as high as the maximum order that we originally computed for cell j. Essentially this boils down to checking a neighborhood around each cell, and letting our solution in the current cell actually use polynomials of an order equal to the largest order in this neighborhood. We only base this on the orders initially computed, and we will initialize these new coefficients to zeros (essentially we are padding vectors with extra zeros). We will refer to this region where we allow the solution to use higher order than is necessary initially as the buffer region. In practice we actually set the buffer region to be one cell wide, and we pick our polynomial re-assignment time to fit that buffer. Call this polynomial re-assignment time Δt_p. We set

$$\Delta t_p = \frac{\Delta x}{\max \lambda(q)}$$

We pick this time because it represents the time at which our solution in cell j could have physically propagated beyond its direct neighborhood. In DG this will still include many timesteps because the linear stability limit is much stricter than the CFL condition.

4.1. **Refining and coarsening in time.** We will need to reallocate local polynomial degrees whenever we expect that high order information should begin propagating outside of our buffer region. Following that we will assign a new buffer. Probably some region in our old buffer still will need high-order and so we will essentially be re-assigning a new buffer around this region. In this way we are mimicking what is often done in AMR where there is a buffer region of very fine cells around the region that actually requires high resolution at any point in time[1]. Our whole algorithm will proceed as follows. An outline of our algorithm is shown in Algorithm 2.

Algorithm 2 Algorithm for p-adaptive DG

```
Compute a projection of the initial conditions
Initialize t_f, dt_p, and t_o = 0.
while t < t_f do
    while t − t_o < dt_p do
        for I_j ∈ I = (a, b) do
            evolve all coefficients
        end for
    end while
    T_o = t
    for I_j ∈ I = (a, b) do
        remove coefficients c_k < ϵ
    end for
    Create buffer regions
end while
```

4.1.1. *Systems.* In systems there are multiple families of coefficients per cell, and in the evaluation of the flux function these different families end up interacting with each other. Currently we are defining one polynomial order per cell, and it is taken to be the maximum of the polynomial order computed for each variable in our hyperbolic system. This is done to handle the case where the evolution of your PDE moves variation between different fields. We want to ensure that we have high enough resolution in every field for this case.

4.2. **Taking advantage of p-adaptivity.** Currently we are only taking advantage of the p-adaptivity by keeping track of a local degree in each cell. We use this to avoid solving unnecessary ODEs on each cell. However we should also really be adapting the quadrature rule used to evaluate

$$\int_{\Omega} \mathbf{F}(\mathbf{q}, \mathbf{x}, \mathbf{t})\phi_x d\Omega$$

Currently one very high-order quadrature rule is being used for all cells, and this is highly over-integrating this term on some cells.

Additionally it should be possible to implement some sort of local time-stepping to take advantage of the fact that the linear stability limit becomes more strict as the polynomial order increases. Thus we should be able to take larger time-steps on cells using low degree polynomials and take smaller timesteps on the cells with higher degree polynomials. This would be a big improvement over letting the worst cell determine a global time-step but it is too complicated to discuss here.

5. Numerical Results

We have implemented our scheme on several simple examples.

5.1. **Advection.**

5.2. **Burgers Equation.**

5.3. **Acoustics.**

References

[1] Marsha J Berger and Randall J LeVeque. Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM Journal on Numerical Analysis*, 35(6):2298–2316, 1998.

[2] Bernardo Cockburn and Chi-Wang Shu. The runge–kutta discontinuous galerkin method for conservation laws v: multidimensional systems. *Journal of Computational Physics*, 141(2):199–224, 1998.

[3] Claes Eskilsson. An hp-adaptive discontinuous galerkin method for shallow water flows. *International Journal for Numerical Methods in Fluids*, 67(11):1605–1623, 2011.

[4] Lloyd N Trefethen. *Approximation theory and approximation practice*. Siam, 2013.

[5] Giovanni Tumolo, Luca Bonaventura, and Marco Restelli. A semi-implicit, semi-lagrangian, p-adaptive discontinuous galerkin method for the shallow water equations. *Journal of Computational Physics*, 232(1):46–67, 2013.