P-ADAPTIVE METHODS FOR HYPERBOLIC PDES

DEVIN LIGHT AND SCOTT MOE

1. Introduction

Numerical methods for solving PDEs are typically developed considering only the simplest of cases. Usually this involves working on problems that can be solved using uiform cartesian grids. However, often hyperbolic PDEs are solved that actually should involve variable amounts of resolution throughout the domain. The CLAWPACK software package addresses this using Adaptive Mesh Refinement (AMR) where a hyperoblic PDE is solved using grids that are more refined in some regions than others. However there is another option that may be just as useful for smooth problems. The Discontinuous Galerkin Finite Element Method (DGFEM) is a high order extension of the Godunov Method. The DGFEM of any order can be implemented on the same grid as the Godunov method. So 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 Discointinuous 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 to some extent. We will develop new strategies that will be useful for p-adaptivley solving hyperbolic PDEs using explicit discretizations. Finally we will examine the performance of these strategies on some one dimensional example problems.

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 method and the Finite Volume method. There are several forms of the DGFEM but I will focus on the formulation that is most similar to the first order Finite Volume method. 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 (we can safely assume it is convex) 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 integral form in equation (**??**).

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

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.

Since we are integrating over the whole domain we have assumed that $\nabla\phi$ is at least integrable over the whole domain. Thus we would have to assume that $\phi \in C_0$.

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 this is is beyond the scope of this document. However the important thing is that we will overcome this issue by making the Finite Element method more like a Finite Volume method. We will use a function space defined locally on cells in the domain, and solve a seperate version of equation (2) on each cell.

For this simple 1-d problem we will split up our domain into subcells $\Omega_k = (x_{k-1/2}, x_{k+1/2})$. Then we will solve 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 more similar to the equations used to derive the first order Godunov method. The main difference comes from the term $\int_{\Omega_k}\mathbf{F}(\mathbf{q},\mathbf{x},\mathbf{t})\phi_x d\Omega_k$ that comes from the fact that our test function is not just a constant (if ϕ is a constant this reduces to the Godunov method).

This is what is known as the Discontinuous Galerkin Finite Element Method.

2.1. **Choosing a Function Space.**

The formulation of the DGFEM presented in equation (3) completely decouples neighbouring cells from one another. So this removes the typical Finite Element method requirement that our basis functions are in C_0. As such we no longer need to pick hat functions, as is typically done. Instead it is normal to pick our basis functions so that on each cell Ω_k we have a heirarchical orthonormal basis. This is the natural extension of the low order Godunov method. Essentially it means that, instead of just evolving the zeroeth 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 linear system which results from terms of the form:

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

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.

As in the Finite Volume methods we are farmiliar with, we will have to define $\mathcal{F}(q_l, q_r)$ However our Riemann problems will involve q_l and q_r that are actually point values evaluated at the interface itself instead of cell averages.

The reasoning for why it is ok to just solve Riemann problems to obtain boundary fluxes is slightly murkier here than in the finite volume case. In this situation you are solving for something in that is really actually very far from a Riemann solution.

3. P-Adaptivity for polynomial approximation

Let us forget about the fact that we actually want to solve a PDE and consider the situation where we are approximating some function $f(x)$ on a line $\Omega = (a, b)$ and we have split our domain into a set of N intervals. Say we just wish to adaptivley approximate f on each one of these intervals. In this case one natural thing to try and do is to try and compute the orthogonal series expansion of our function.

Let us define $L_i(x)$ 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

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

We know from approximation theory that these coefficients should decay at some rate depending on the local regularity of f. If we want to mantain a certain accuracy for our specific cell j then we should expect to only need a finite number of these coefficients. Clearly if we want a this accuracy on each cell, the number of important coefficients on each cell could be very different. The most obvious case of this is when f is constant on a cell. In this case we should only need one coefficient.

The idea of p-adaptive approximation is to allow this order to change naturally from cell to cell. This could 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.

Algorithm 1 Algorithm for p-adaptive approximation

```
1:  for $I_j \in I = (a, b)$ do
2:      $c^n = \int_{I_j} f(\xi) L_0(\xi) d\xi$
3:      $c^o = c^n$
4:      $k = 0$ and $c^j_k = c^n$
5:      while $\max(|c^o|, |c^n|) > TOL$ and $k < N_m - 1$ do
6:          $c^o = c^n$
7:          $k = k + 1$
8:          $c^n = \int_{I_j} f(\xi) L_k(\xi) d\xi$
9:          $c^j_k = c^n$
10:     end while
11: end for
```

4. P-Adaptivity for A Discontinuous Galerkin Scheme

This is what we are currently working out. We hope to use Algorithm 1 to compute an initial condition. After that we will go through and reassign orders on each cell to create a buffer region around each of the high order regions. The idea behind this is that we will not need to worry about reassigning new orders every time-step.

4.1. **Refining and Coarsening in time.**

The idea is that every handfull of timesteps we should coarsen everywhere possible, then reassign a new buffer. Hopefully 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 only.

Here we will, hopefully have some nice figures from like the advection equation where we have high order cells just following the interesting region of some profile. We need to get the code working for this.

5. Examples

We will show a couple examples of following profiles of differing regularities. We have been working on making our code general so that it should work for any of these examples....

5.1. **Advection.**

5.2. **Acoustics.**

5.3. **Shallow Water.**