# Discontinuous Galerkin Methods and their applications

Bernat Guillen

May 18, 2015

bernatp@princeton.edu

# Contents

# 1 Introduction

In this report I will introduce the general concepts and practical implementation issues of discontinuous Galerkin (dG) methods. These are a family of methods that try to achieve the flexibility of Finite Element Methods (FEM) and the good properties of Finite Volume Methods (FVM). In this section we will see the reason to study different methods for solving the same problem, and the differences between the most commonly used methods, giving a motivation for the dG methods. In section 2 we will cover the definition of dG methods as well as some important theoretical results. In section 3 we will cover the details of the 1D implementation in C++. In section 4 we will see the implementation in MATLAB of 2D dG methods. Finally, in section 5 we will comment on the viability of using dG methods in scientific applications such as MHD. All the work and MATLAB code has been based off [1], which has proven to be invaluable in learning the basics and the not-so-basics of dG methods.

## 1.1 Motivation: From Finite Difference to dG methods

We start off with a PDE of the form 1, where $\mathcal{H}$ is a function space.

$$\partial_t u + \partial_x f(u(x,t),x,t) = g(x,t),\ x \in \Omega,\ u \in \mathcal{H} \tag{1}$$

The PDE could be more general but for the purpose of this section we will focus on conservation laws in one dimension. We want to approximate the solution to this PDE in a (possibly) different space $\mathcal{H}_h$ in a way that is computable and fast enough.

The first approach that comes to mind is Finite Difference methods, where we consider all the functions smooth enough and we discretize the domain $\Omega \subset \mathbb{R}$ in $x^1, \dots, x^K$. We have a new equation approximated, for example, by 2 where $h^k = x^{k+1} - x^k$.

$$\frac{du_h(x^k,t)}{dt} + \frac{f_h(x^{k+1},t) - f_h(x^k,t)}{h^k + h^{k-1}} = g(x^k,t) \tag{2}$$

Note that we require that, in the neighborhood of each grid point, $u_h$ and $f_h$ are well approximated by local polynomials, and thus $u_h(x,t) = \sum_{i=0}^{2} a_i(t)(x - x^k)^i$ for $x \in [x^{k-1}, x^{k+1}]$, and similarly for $f_h$, in a way that interpolates $u$ and $f$ in the grid points. In more dimensions, we would need a dimension-by-dimension structure of the grid. The residual in this case is

$$\mathcal{R}_h(x,t) = \partial_t u_h(x,t) - \partial_x f_h(u_h(x,t),x,t) - g(x,t) \tag{3}$$

If $\mathcal{R}_h(x,t)$ was 0 everywhere, then $u_h$ would solve the equation. Thus, it is not and we have to define in which way are we asking $u_h$ to follow the equation. In this case, we ask that $\mathcal{R}_h(x^k,t) = 0$ for all the grid points, which is equivalent to defining Eq. 2.

This approach is very simple and direct, also gives flexibility in the timestepping, although it requires more elaborate strategies for boundaries or discontinuities and requires more structure for higher dimensions.

This discussion suggests that a more general approach than one-dimensional approximation via polynomials is required to ensure geometric flexibility. A first alternative comes from the FVM. In the FVM we divide our domain $\Omega$ in elements $\mathbf{D}^k$ and (in

its simplest form) approximate $u(x,t)$ for $x \in \mathbf{D}^k$ by $\overline{u}^k(t) = \int_{\mathbf{D}^k} u(x,t)dx$. The residual is now $\mathcal{R}_h(x,t) = \partial_t \overline{u}^k(t) + \partial_x f - g(x,t)$ for $x \in \mathbf{D}^k$. If we require that $\int_{\mathbf{D}^k} \mathcal{R}_h(x,t)dx = 0$, then we arrive to the basic FVM. This approximation is purely local and does not require conditions on the grid structure. However, note that we need the values of $f$ at the boundaries of the cells, and these values depend on $u$, and thus a flux reconstruction strategy is needed as we have seen during the course. Furthermore, if we need a bigger order, we will have to ask more information from the surrounding cells, losing in the process some locality.

Following with this elements approach, but adding more degrees of freedom in each cell (to be able to get more accuracy without losing locality), we get the FEM. Let us assume we have the elements $\mathbf{D}^k$ defined by $[x^k, x^{k+1}]$ with $K$ elements and $K+1$ grid points. We assume that the local solution in this element is

$$u_h(x) = \sum_{i=1}^{N_p} b_n \psi_n(x) \tag{4}$$

where $\psi_n(x)$ is a locally defined basis function, for example linear functions in $\mathbf{D}^k$. We can write then, for $x \in \Omega$

$$u_h(x,t) = \sum_{k=1}^{K} u(x^k,t)N(x) = \sum_{k=1}^{K} u^k(t)N^k(x) \tag{5}$$

Where $N^i(x^j) = \delta_{ij}$. To recover the scheme to solve Eq. 1 we ask that the residual is orthogonal to all test functions in a space $\mathbf{V}^h$. That is

$$\int_\Omega (\partial_t u_h + \partial_x f_h - g_h)\phi_h(x)dx = 0 \; \forall \phi_h \in \mathbf{V}^h \tag{6}$$

If we choose $\mathbf{V}^h$ equal to the search space of solutions, this is the same as saying $(\mathcal{R}_h, N^j) = 0$ for all $j$ (Note: $(,)$ is the scalar product). This is called a Galerkin scheme. If we consider the vectors $\mathbf{u}_h = (u(x^1,t), \ldots, u(x^{K+1},t))$, $\mathbf{f}_h = (f(u(x^1,t),x^1,t), \ldots, f(u(x^{K+1}$ and $\mathbf{g}_h = (g(x^1,t), \ldots, g(x^{K+1},t))$, and the matrices $M, S$ defined by $M_{ij} = (N^i, N^j)$, $S_{ij} = (N^i, N^{j'})$, then the equation gets reduced to:

$$M\frac{d\mathbf{u}_h}{dt} + S\mathbf{f}_h = M\mathbf{g}_h \tag{7}$$

The problem with this approach is that $M$ is a global matrix and we have to invert it. If the number of elements is big, this will be costly.

The next step is therefore not requiring any kind of continuity in the grid points. This means that we can treat the problem as having double the number of nodes, but the resulting matrices will be easy to invert (will be local and only depend on $N_p$). If we define in each element $\mathbf{D}^k$ $u_h^k(x,t) = \sum_{i=1}^{N_p} \overline{u}_h^{i,k}(t)\psi_h^{i,k}(x)$ then we actually have $u_h^k(x^{k+1},t) \neq u_h^{k+1}(x^{k+1},t)$. This allows us to define $\mathcal{R}_h(x,t) = \partial_t u_h^k + \partial_x f_h^k - g(x,t)$ for $x \in \mathbf{D}^k$. Here the space of test functions is again the same as the space of basis functions, defined as $V_h = \bigoplus_{k=1}^K \{\psi_i^k\}_{i=1}^{N_p}$. Thus we require that the residual is orthogonal to $\psi_i^k$ for all the base functions $(k,i)$.

Integrating by parts we obtain the following:

$$\int_{\mathbf{D}^k} (\partial_t u_h^k)\psi_j^k - f_h^k \partial_x \psi_j^k - g\psi_j^k dx = -[f_h^k \psi_j^k]_{x^k}^{x^{k+1}} \tag{8}$$

The problem, as it happens with FVM, is that $f_h^k(x^k)$ is not totally well defined and it depends on both nodes $\mathbf{D}^{k,k-1}$. As with FVM, we need to define the numerical flux in the boundaries, $f^*$. Integrating by parts again we obtain the formulation for the weak solution (Eq. 9) and the formulation for the strong solution (Eq. 10).

$$\int_{\mathbf{D}^k} (\partial_t u_h^k)\psi_j^k - f_h^k \partial_x \psi_j^k - g\psi_j^k dx = -[f^* \psi_j^k]_{x^k}^{x^{k+1}} \tag{9}$$

$$\int_{\mathbf{D}^k} \mathcal{R}_h(x,t)\psi_j^k dx = [(f_h^k - f^*)\psi_j^k]_{x^k}^{x^{k+1}} \tag{10}$$

Again, we can obtain matrices $M$ and $S$ in the same way. Note that they are sparser now, although they will grow with the dimension of the base in each element. There will be an overhead created by "'counting twice"' each node, although this overhead will decrease when the number of elements grows.

# 2 Theory of dG methods

We analyze in more detail the method and provide some results on convergence and error sources. The proofs can be seen in more detail in [1].

The choice of the elements of the base allows us to distinguish between nodal and modal dG methods. If we denote $l_{i,h}^k$ the polynomials in $\mathbf{D}^k$ that interpolate the points $x_i^k \in \mathbf{D}^k$, we can say that

$$\sum_{i=1}^{N_p} \overline{u}_i^k(t)\psi_i^k(x) = \sum_{i=1}^{N_p} u_i^k(t) l_{i,h}^k(x) \tag{11}$$

In this case, $\overline{u}_i^k(t)$ are the modes of the solution and $u_i^k(t) = u_h(x_i^k, t)$ are the nodes of the solution. Although conceptually both definitions are equivalent, computationally they are different. Let us take normalized Gauss-Jacobi polynomials $P_{i-1}$ (Legendre polynomial of degree $i-1$) as the base $\psi_i^k$ transformed from the polynomials defined in $[-1,1]$, and Lagrange polynomials interpolating on Gauss-Lobatto points ($r_i$ in $[-1,1]$) as the base $l_i^k$. Note that the Gauss-Lobatto quadrature points come from the Gauss-Jacobi polynomials, so this is a practical consideration that will have more advantages but we will study it in next section. In order to transform from $\overline{u}_i^k$ to $u_i^k$ (on $[-1,1]$) we simply have to use the matrix $V$ where $V_{ij} = P_j(r_i)$ ($k$ is not necessary here). Note that $u(r_i) = \sum_{n=1}^{N_p} \overline{u}_n P_{n-1}(r_i)$ so $V\overline{u} = u$. This matrix is a generalized Vandermonde matrix and connects nodal and modal approaches. Similarly, since $l_i$ are Lagrange polynomials and due to the uniqueness of the interpolation, $P_j(r) = \sum_{i=1}^{N_p} P_j(r_i) l_i(r)$ or $P(r) = V^T l(r)$.

Remembering the definitions of $M^k$ and $S^k$, we have $M_{ij}^k = (l_i^k, l_j^k) = \frac{h^k}{2}(l_i, l_j)$. Remember $h^k$ is the size of the element $k$, and $l_i, l_j$ are defined in $[-1,1] = I$. Now,

using our definition of $V$, we have that $(l_i, l_j) = (((V^T)^{-1}P)_i, ((V^T)^{-1}P)_j)$. From here we have

$$(l_i, l_j) = \sum_{n=1}^{N_p}\sum_{m=1}^{N_p}(V^T)^{-1}_{in}(V^T)^{-1}_{jm}(P_{n-1}, P_{m-1}) = \sum_{n=1}^{N_p}(V^T)^{-1}_{in}(V^T)^{-1}_{jn} \tag{12}$$

In conclusion, $M^k = \frac{h^k}{2}(VV^T)^{-1}$. Similarly $S^k_{ij} = (l^k_i, \partial_x l^k_j)$ and if we define $D_{r,(i,j)} = \partial_x l_j(r_i)$ then $S = MD_r$.

Let us now study the convergence of the method by considering the linear hyperbolic system

$$\partial_t u + A\partial_x u = 0 \tag{13}$$

where $A$ is diagonalizable and we assume that the problem is well-posed, i.e. there exists constants $C$ and $\alpha$ such that $||u(t)||_\Omega \leq Ce^{\alpha t}||u(0)||_\Omega$. Let us assume that we approximate the solution by a $N$th order piecewise polynomial, $u_h$ (denoting the vector of values), which satisfies

$$\partial_t u_h + L_h u_h = 0 \tag{14}$$

Here, $L_h$ represents the discrete approximation of $A\partial_x$. The truncation error is $\partial_t u + L_h u = T(u(x,t))$. If we define $\epsilon(x,t) = u(x,t) - u_h(x,t)$ we seek convergence in the sense that $\forall t < T$, $\lim_{\text{dof}\to\infty}||\epsilon(t)||_{\Omega,h} \to 0$. Convergence can be achieved either by decreasing the cell size ($h$) or the order $N$ of approximation, or both simultaneously (hp-convergence) hence the term "'degrees of freedom"' (dof). If we study

$$\partial_t \epsilon + L_h \epsilon = T(u(x,t)) \tag{15}$$

we see we can solve it by

$$\epsilon(t) - e^{-L_h t}\epsilon(0) = \int_0^t e^{L_h(s-t)}T(u(s))ds \tag{16}$$

Integrating over the elements and summing it, we get

$$||\epsilon(t)||_{\Omega,h} \leq ||e^{-L_h t}\epsilon(0)||_{\Omega,h} + ||\int_0^t e^{L_h(s-t)}T(u(s))ds||_{\Omega,h} \tag{17}$$

It suffices to show consistency in the form of $||\epsilon(0)|| \to 0$ and $||T(u(t))|| \to 0$ as the degrees of freedom tend to infinity, and stability as $||e^{-L_h t}|| \to A(t) \leq C_h e^{\alpha t}$ to guarantee convergence.

First of all let us focus on the consistency in the local element (order refinement). For simplicity, $h^k = h$ and all elements have the same size. The error of interpolating $v \in H^p(I)$, $p > 1/2$ with $v_h$ of order $N$, for $0 \leq q \leq p$ is

$$||v - v_h||_{I,q} \leq N^{2q-p+1/2}|v|_{I,p} \tag{18}$$

The result is slightly different for the modal approach (but we only lose at most one order of convergence).

The main result for consistency, where we focus both on $h$ and $N$, is that, for $u_h$ a piecewise polynomial approximation of order $N$ of $u$ in $H^p(\mathbf{D}^k)$, $p > 1/2$ we have

$$||u - u_h||_{\Omega,q,h} \leq C \frac{h^{\sigma-q}}{N^{p-2q-1/2}} |u|_{\Omega,\sigma,h} \tag{19}$$

where $1 \leq q \leq \sigma$ and $\sigma = \min(N+1, p)$.

Stability, on the other hand, is derived from the definition of flux. It can be proved that if the problem is hyperbolic (i.e. $A$ is uniformly diagonalizable) then we have stability.

Using these results we can arrive at a bound on the error of

$$||\epsilon_h(T)|| \leq (C_1 + C_2 T) h^{N+1/2} \tag{20}$$

in the limit where $p \gg N+1$ and $u \in H^p(\Omega)$.

We can establish also boundaries on the dispersive error, dissipative error and phase error, which decay much faster than the accuracy error if we increase the number of elements, and exponentially (with some caveats) if we increase the order.

Finally, a CFL condition can be found and $\Delta t \leq \frac{C}{\max\{|lambda(A)|\}} \min_{k,i} \frac{h^k}{2} (\Delta_i r)$, where $C$ depends on the timestepping. More interestingly, $\Delta t = O(\frac{h}{N^2})$.

This concludes the theoretical analysis of dG methods and the presentation of some interesting results on convergence and accuracy.

# 3 Implementation in 1 Dimension

The implementation in 1 Dimension has been done using C++. The code, and some examples, can be seen in `github.com/bernatguillen/dG_project`. The code consists of a class `dG1D_Framework` which contains all the matrices, vectors and "global" variables of the solver, and a library for each problem solved (advection, advection with discontinuity, and Maxwell's equations). Unfortunately, some bug that I have not been able to identify is causing problems with Euler compressible gas dynamics and I have resolved to using MATLAB instead to show how well-behaved dG methods are for shocks after filtering and antialiasing. Furthermore, all the code done in MATLAB is not central to the dG methods but required for the examples. This means that if we did not take into account discontinuities and mesh generation in higher dimensions, the code done in C++ would be enough. However, doing this project I have found that most of the work that dG methods requires is not related to the method itself but to the details that make it really useful.

The only data `dG1D_Framework` needs is a list of nodes, and the matrix connecting those nodes, as well as the degree of the polynomials in each element. When `dG1D_Framework` is declared, it creates the Gauss-Lobatto points, generalized Vandermonde matrix, its inverse, $D_r$, and some matrices that indicate the connections between faces and faces, or elements and faces. The Gauss-Lobatto points are used so that they maximize the determinant of $V$ (the generalized Vandermonde matrix), thus making it better conditioned for inverting. The matrix of connections is used to create a matrix LIFT which is what will be used to calculate the flux in each element boundary with the appropriate signs. The code also generates a 1 dimensional grid. It requires GSL library (for inverting the matrix $V$), and is compiled using "make all". In order to

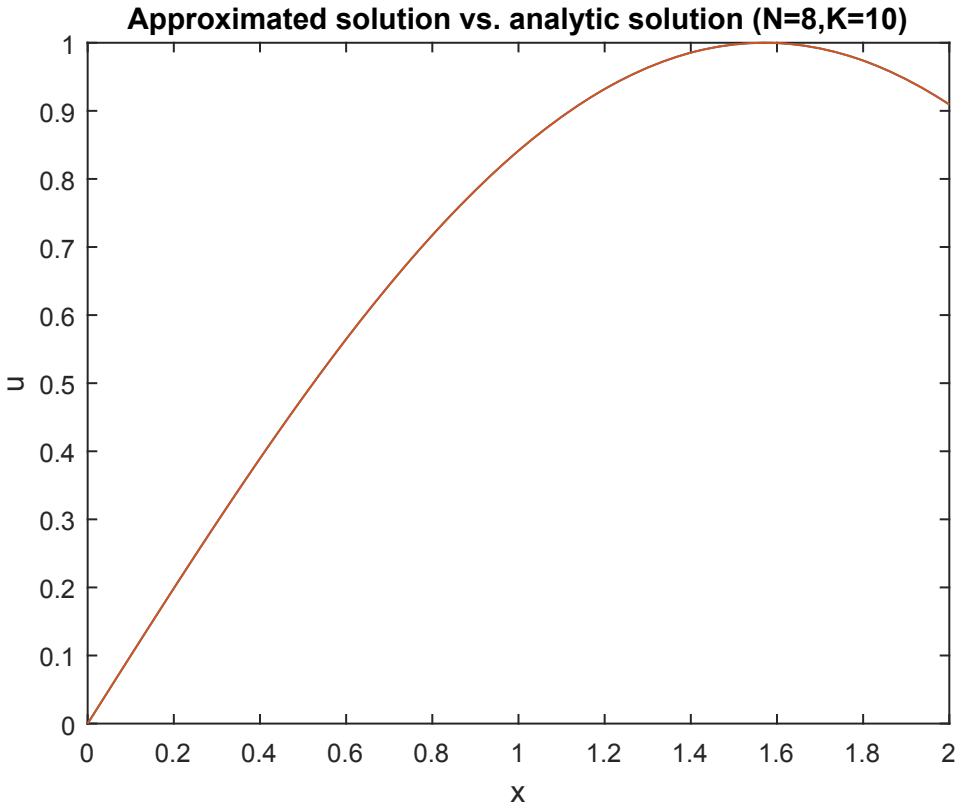


**Figure 1:** $u = \sin(x - 2\pi t)$ for $t = 10$

run, execute `Adv_1D` for 1D advection (inflow), `Adv_disc` for discontinuous 1D advection (periodical boundary conditions), `Maxwell_1D` for Maxwell's equations, and `unittest` for testing the parts of the declaration. The code for the generation of the method is in `mesh1d.{c,h}`. The fluxes in each example are defined according to [1], i.e. using averages and differences of the values of $u_h$ in each of the elements (for the face).

In Fig. 1 we can see the real solution and the approximated solution overlapping each other. In Fig. 2 and 3 we see how does the error decrease or grow when the order of approximation or the number of elements grows. Note the sawtooth pattern for even orders. That is due to the choice of the flux (it has a centered component instead of only upwind). Note also it decreases roughly as $O(h^{N+1/2})$ although it quickly reaches machine precision.

In Fig. 4 we see how do discontinuities affect the solution. Namely, this is a Gibbs effect caused by the high frequencies present in the discontinuity (basically, $p$ such that $u \in H^p$ is not big enough compared to $N$. It is solved by defining a filter, based on a function $\sigma(\eta)$, such that $\sigma(0) = 0$, $\sigma(\eta) = 1$ for $\eta > 1$ and $\sigma(\eta) = 1 - \alpha\eta^{2\hat{s}}$ elsewhere. Multiplying each mode $\overline{u}_n^k$ by $\sigma(\frac{(n-1)}{N})$ results in an order $\hat{s}$ filter. The parameters have to be chosen arbitrarily, finding the right bandwidth of the filter is not easy. In Fig. 5 we see a simulation of Sod's problem (Euler equations), after filtering (and slope limiting) has been applied (code in MATLAB).

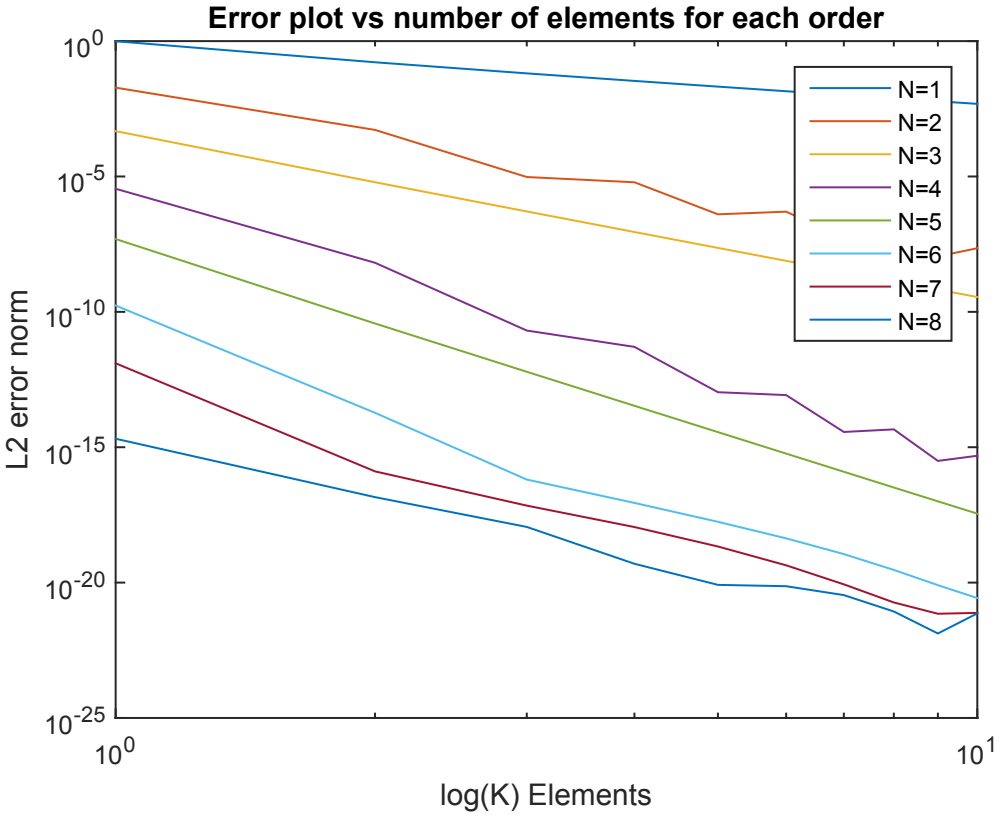


**Figure 2:** Error reduction when we increase the number of elements, for $N$ fixed

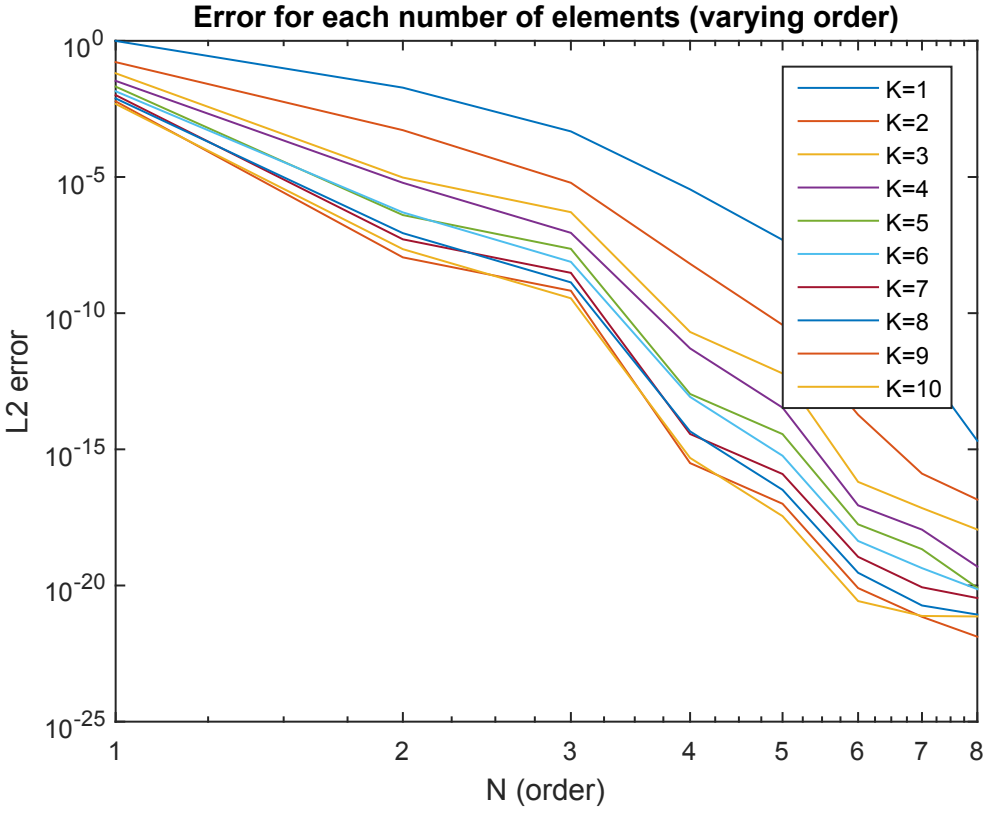


**Figure 3:** Error reduction when we increase the order of approximation, for fixed $K$

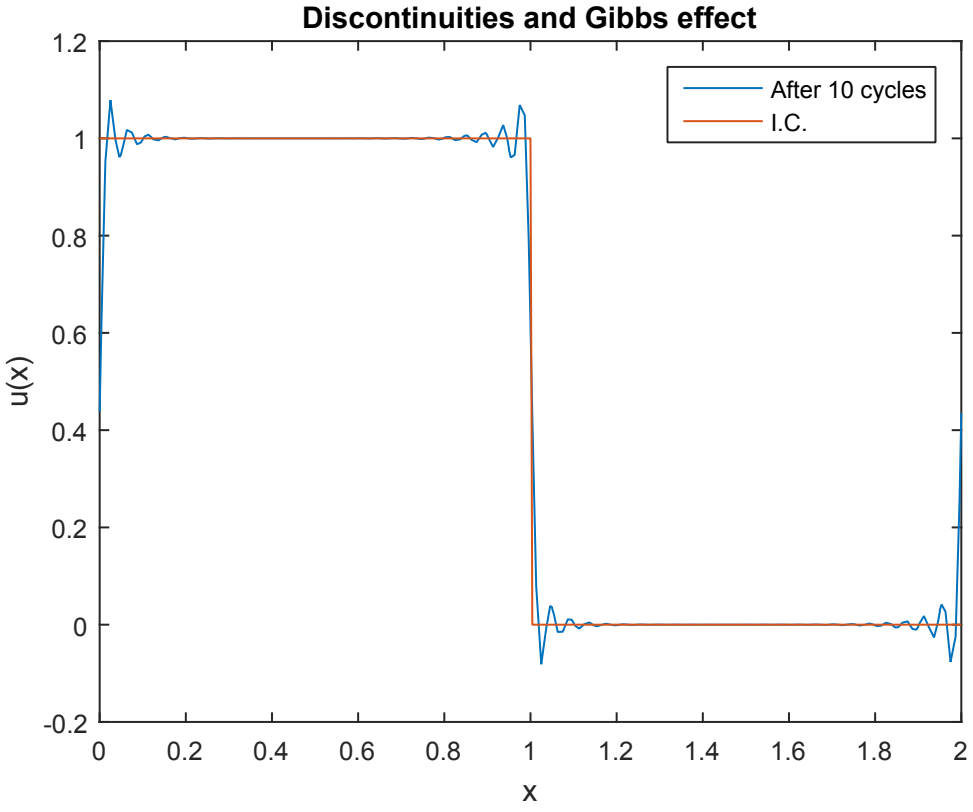


**Figure 4:** See the Gibbs phenomenon in the discontinuities

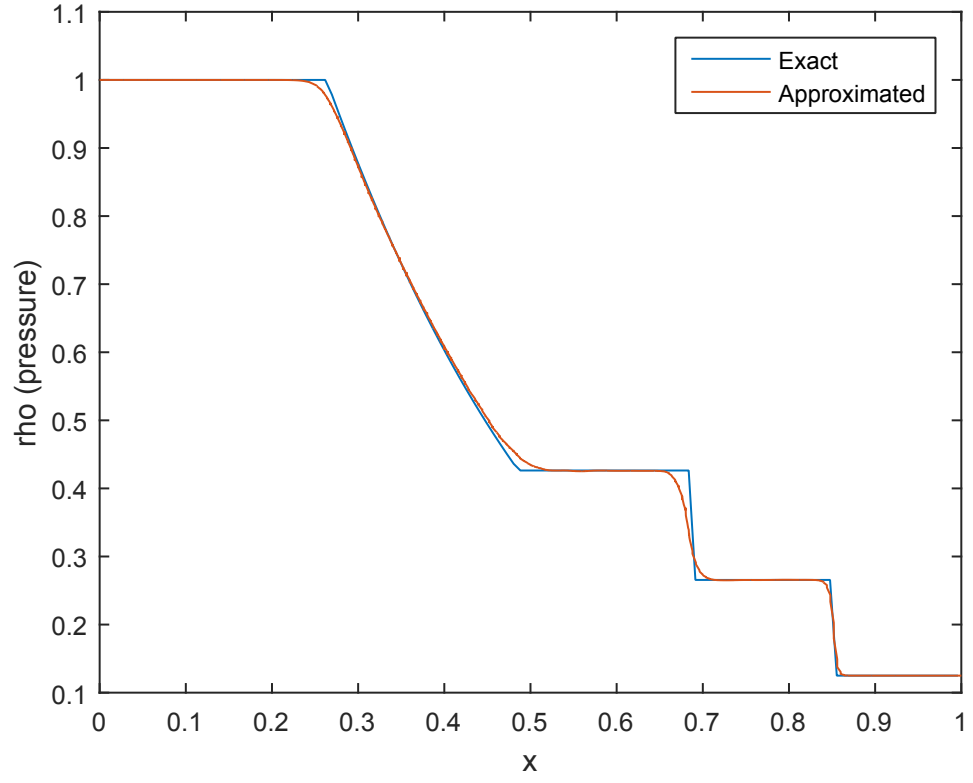


**Figure 5:** Sod's tube shock problem, computed density after 0.2s. K=250, N=6

# 4 2 Dimensions, issues

For the 2 dimensions, I used MATLAB. The method itself is exactly the same, only with bigger matrices that show the connectivity between nodes and faces. The problem is that generating a mesh is already a significant amount of work and my options were either generate the mesh or learn how to read files in a specific format (which I could read seamlessly in MATLAB). I decided that for studying the properties of dG methods it was better to use MATLAB. The MATLAB codes are available in [1] and `http://www.nudg.org/` and I used those (changing a few things to make the plots), so I have not included them in my git repository.

As said, the theoretical properties of dG methods are unchanged, although the choice of the base of the test functions and the points used in the Nodal dG method is a bit different. Using a triangular mesh and a simple tensor product of the Gauss-Lobatto points would clusterize all the points near the vertexes of the simplex. Thus it is necessary to warp the nodal points inside the simplex and make it more evenly distributed. This reference simplex is then translated into the element by an affine transformation.

The images generated by the 2D code are too big to be rendered in this pdf, so I will include them as annexes. The file `maxwell.pdf` shows the solution to Maxwell's equation in a metallic air-filled cavity $\Omega = [-1, 1]^2$, with perfectly electrically conducting walls. It can be seen [1] that the error converges as $O(h^{N+1})$.

The files `eulershock.pdf` and `eulershock3d.pdf` show a 2D forward facing step simulation with fluid entering at Mach 3 from the left. This result cannot be compared analytically with anything, but comparison with previous results gives confidence in it.

# 5 Conclusions and Future Work

Discontinuous Galerkin methods are very powerful and a viable alternative to FVM for conservation laws. They are flexible and allow for great hp-convergence in both number of elements and order of approximation. As more precision is required in simulations and more computational power can go into them, I expect these methods to gain popularity amongst not only the CFD community but also any kind of problems that require working on different scales in both time and space. Unfortunately, the flexibility of the method is only paralleled by the overhead in development times that it requires in order to solve the issues that can come from discontinuities or mesh generation. It is understandable however and one has to ponder all costs and benefits before using a new method.

One of the big advantages of dG methods is that changing the order of convergence does not require having a completely different scheme (for example getting a pentadiagonal matrix instead of tridiagonal, or using values in nodes that are further away), and once the first implementation is done, the only problem is finding a good numerical flux. Its good properties in dispersivity and diffusivity make it also a good choice.

The fact that dG methods are not required to have continuity in the boundaries of the elements does not imply any problem (except that some filtering is required, but that is still better than not being able to simulate it). On the contrary, convergence in Sobolev

spaces is guaranteed as long as the solution is sufficiently smooth.

The computational cost of the method is small, having to invert only once a matrix that will usually have at most a couple dozen columns and rows. The overhead of the initial computations is easily compensated by the rest of the process.

dG methods can be straightforwardly parallelized, requiring communication between each element in the same way FVM do, to compute the fluxes. This would reduce the computation with a good domain decomposition.

The original idea was to compare these results to Athena's and that would be the next step, however unfortunately I have not had time to make Athena work in my workstation (due to permissions mainly). Nevertheless I think that at least all the 1D code has been tested against analytic solutions and the 2D code both against analytic solutions and other simulations and performed well. There have been recent results on dG methods for MHD that might be relevant [2].

The code has been profiled with gprof and version controlled with git. I used gdb for the debugging process and gsl for matrix inversion. Unfortunately this made it impossible to test my code in the workstation (again, permissions) and so all the tests had to be done in my laptop, which means that not much can be said about performance since the problems have not been that big. My C++ code has run always in less than 3 seconds, even for large integration times and lots of elements and high degrees of the polynomials.

I intend to continue using dG methods for other problems (maybe for numerical relativity) and compare them with traditional Finite Differences or Finite Volumes methods. The first step is generating meshes in 2,3 and 4 dimensions, which is already a big challenge. Then the code will be ported to C++ to gain performance. The code will be maintained in `https://github.com/bernatguillen/dG_project`.

# References

[1] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[2] O. Zanotti, F. Fambri, and M. Dumbser. Solving the relativistic magnetohydrodynamics equations with ADER discontinuous Galerkin methods, a posteriori subcell limiting and adaptive mesh refinement. *ArXiv e-prints*, April 2015.