Outline

• Introduction
  – Hyperbolic Conservation Laws
  – Numerical Methods for PDEs

• The Discontinuous Galerkin (DG) Method
  – Introduction in 1D
  – Extension to higher order and higher dimensions
  – Parallel Computation

• Summary and Further Reading
Introduction

1. Continuous problem
2. Impose spatial discretization
3. Approximate solution on each cell
4. Compute fluxes across cell interfaces
5. Evolve solution in time
6. (Do all this as efficiently as possible)
The DG Method – Short History

• DG was first proposed by Reed and Hill in 1973
  \[ \sigma u + \nabla \times (au) = f \]

• First analysis (LeSaint and Raviart, 1974) showing order \( k \) accurate in general and often \( k + 1 \) accurate for smooth solutions (\( k \) polynomial degree of piecewise approximation)

• Sharp analysis (Johnson, 1986) showed \( k + \frac{1}{2} \)-order accurate

• However, the schemes did not enjoy much use early on
The DG Method – Short History

• Extension from scalar conservation laws in late 1980s to systems in late 1990s (Cockburn and Shu)

• Development of limiters and Runge-Kutta DG methods for nonlinear conservation laws

• New application areas, e.g., Maxwell’s equations, magneto-hydrodynamics, water waves, and elasticity
The DG Method – Short History

The last decade has seen an explosion in activities
- Higher order problems
- Hamilton-Jacobi equations
- Non-coercive problems and spectral accuracy
- Adaptive solution techniques
- Improved solvers
- Advanced time-integration methods
- Large scale production codes
- Etc.
Classes of PDEs

Elliptic

- Infinite speed of propagation
- All cells coupled
- Must assemble and solve a linear system
- Prototypical example: Laplace’s Equation

Parabolic

- Typically describe diffusion phenomena
- Can use explicit time stepping methods
- Prototypical example: The Heat Equation
Hyperbolic Conservation Laws

- Fundamental in nature
- Conservation laws arise in a multitude of different scientific areas
- For example:
  - Magneto-hydrodynamics
  - Elastic waves
  - Acoustic waves
  - Gravitational waves
  - Fluid dynamics (Euler eq’s, etc.)

- Discontinuities propagate without being smoothed, e.g., shock waves
- Solution propagates with a finite speed
  - Enables explicit time-stepping schemes
  - Finite domains of influence and dependence
Hyperbolic Conservation Laws

A conservation law states that the sum of a particular quantity within an isolated system does not change as the system evolves in time, e.g., mass, momentum, and energy.

A general nonlinear hyperbolic conservation law in 1D can be written in differential form as $u_t + F(u)_x = S(u)$,

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad F(u) = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}, \quad S(u) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{bmatrix},$$

in which $u(x, t)$ is a vector of conserved variables, $F$ is the flux vector, and $S$ is a source vector.

Solutions generally defined in a weak sense only
Hyperbolic Conservation Laws

• The Shallow Water Equations (written in 1D)

\[
\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} = 0 \quad \text{(Mass)}
\]
\[
\frac{\partial hu}{\partial t} + \frac{\partial (hu^2 + \frac{1}{2}gh^2)}{\partial x} = 0 \quad \text{(Momentum)}
\]

Fresnel visualization (2D)
Hyperbolic Conservation Laws

- The Euler Equations of compressible gas dynamics (written in 1D)

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} &= 0 \\
\frac{\partial \rho u}{\partial t} + \frac{\partial (\rho u^2 + p)}{\partial x} &= 0 \\
\frac{\partial E}{\partial t} + \frac{\partial (E + p)u}{\partial x} &= 0
\end{align*}
\]

(Mass)  
(Momentum)  
(Energy)

\[p = (\gamma - 1) \left( E - \frac{1}{2} \rho u^2 \right), \quad c = \sqrt{\frac{\gamma p}{\rho}}\]

(Ideal gas law)

Schlieren visualization (2D)
Numerically Solving PDEs

In general, we cannot find an analytical solution

In the following we consider the 1D scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = s, \quad x \in \Omega,$$

in which $f(u)$ is the flux function and $s(x, t)$ is a source function

We will now briefly investigate three classical methods for solving this PDE numerically and list some benefits and problems associated with each method
Numerically Solving PDEs

• To construct a numerical method for solving PDEs we need to consider
  – How to represent the solution $u(x, t)$ by an approximate solution $u_h(x, t)$?
  – In which sense will the approximate solution $u_h(x, t)$ satisfy the PDE?

• These two choices separate and define the properties of different numerical methods

• Bottom line is that we need ways to
  – Generate a system of algebraic equations from the well-posed PDE
  – Incorporate boundary conditions
  – Solve the system of equations while minimizing unavoidable errors that are introduced in the process
Finite Difference Methods (FDM)

- Domain is represented by a set of points
- Solution approximated by function values at these points
- The equation is satisfied in a pointwise manner

\[
\frac{du_h(x_j, t)}{dt} + \frac{f_h(x_{j+1}, t) - f_h(x_{j-1}, t)}{h_j + h_{j-1}} = s(x_j, t)
\]
Finite Difference Methods (FDM)

• Main benefits
  – Simple to understand
  – Straightforward implementation on structured meshes
  – Method is local and can be made explicit in time
  – Simple techniques for local adaptivity (upwinding)
  – Extensive body of theoretical and practical work on these methods since the 1960s

• Main problems
  – Implementation complexity increases for irregularly shaped geometry
  – Less well-suited for problems with discontinuities
  – Grid smoothness requirements
Finite Volume Methods (FVM)

• Domain is represented by non-overlapping cells
• The local approximation is a cell average

\[ \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_h(x)dx = h_j \bar{u}_j \]

• The equation is satisfied on conservation form

\[ h_j \frac{d\bar{u}_j}{dt} + f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}} = h_j \bar{s}_j \]
Finite Volume Methods (FVM)

- Not straightforward to evaluate the fluxes on cell interfaces
- Numerical flux functions are generally used:

\[ f_{j+\frac{1}{2}} = F(u_{j+\frac{1}{2}}^-, u_{j+\frac{1}{2}}^+) \]
Finite Volume Methods (FVM)

• Main benefits
  – Robust and fast due to locality
  – Support resolution of complex geometries
  – Well-suited for hyperbolic conservation laws (local upwinding)
  – Method is local and can be made explicit in time
  – Method is locally conservative
  – Extensive theoretical framework since the 1970s

• Main problems
  – Difficult to achieve high-order accuracy in a straightforward way on general grids due to requirement for extended stencils (reconstruction problem)
  – Grid smoothness requirements
Finite Element Methods (FEM)

- Domain is represented by non-overlapping elements
- The solution is defined in a non-local manner, using piecewise continuous polynomials:
  \[ u_h(x) = \sum_{j=1}^{J} u(x_j)N^j(x) \]
- The equation is satisfied globally:
  \[ \int_{\Omega} \left( \frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - s_h \right) N^k(x) dx = 0 \]
Finite Element Methods (FEM)

• Main benefits
  – Robust
  – Systematic implementation on unstructured meshes
  – High-order accuracy can be combined with complex geometries
  – Well-suited for elliptic problems (global statement)
  – Extensive theoretical framework since the 1970s

• Main problems
  – Standard Galerkin not well-suited for problems with direction (global statement), but there are techniques to handle such problems, like Petrov-Galerkin, streamline diffusion, etc.
### Summary of Numerical Methods

<table>
<thead>
<tr>
<th></th>
<th>Complex geometries</th>
<th>High-order accuracy and $hp$-adaptivity</th>
<th>Explicit semi-discrete form</th>
<th>Conservation laws</th>
<th>Elliptic problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDM</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FVM</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FEM</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DG</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Want a scheme that combines:
  - The high-order element of FEM
  - The geometric flexibility of FEM and FVM
  - The local statement on the equation for FVM

The Discontinuous Galerkin Finite Element Method
The Discontinuous Galerkin Method

We start with a 1D homogeneous scalar conservation law

\[ u_t + f(u)_x = 0 \]

Multiply with a test function \( v \) and integrate over some cell \( I_j = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}] \)

\[ \int_{I_j} u_t v + f(u)_x v \, dx = 0 \]

Then integrate by parts

\[
\int_{I_j} u_t v \, dx - \int_{I_j} f(u) v_x \, dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]
The Discontinuous Galerkin Method

Next, we assume that both the solution $u$ and the test function $v$ comes from a finite dimensional approximation space $V_h$, usually the space of piecewise polynomials of degree up to $k$:

$$V_h = \{ v : v|_{I_j} \in P^k(I_j), \quad j = 1, \ldots, N \}$$

For example $P^2 = \{1, x, x^2\}$

Assume unit cell to simplify the further derivation
Example: DG(0)

Let us start with the simple basis of \( P^0 = \{1\} \):

\[
\int_{l_j} u_t v dx - \int_{l_j} f(u) v_x dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

\[
\int_{l_j} u_t 1 dx - \int_{l_j} f(u) 0 dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

\[
\bar{u}_j \int_{l_j} u_t dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

This is nothing else than the finite volume method

\[
f_{j+\frac{1}{2}} = F(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}})
\]
Example: DG(1)

We try again, using a different basis, $P^1 = \{1, x\}$:

\[
\int_{I_j} u_t v dx - \int_{I_j} f(u) v_x dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

$v = 1$

\[
\int_{I_j} u_t dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

$v = x$

\[
\int_{I_j} u_t x dx - \int_{I_j} f(u) dx + f(u_{j+\frac{1}{2}}) v_{j+\frac{1}{2}} - f(u_{j-\frac{1}{2}}) v_{j-\frac{1}{2}} = 0
\]

New term
Example: DG(1)

By rearranging and inserting

\[ u(x, t) = a_0(t) \cdot 1 + a_1(t) \cdot x \]

we get

\[ v = 1 \]

\[ \frac{da_0}{dt} \int_{l_j} 1 \, dx + \frac{da_1}{dt} \int_{l_j} x \, dx = -f(u_{j+\frac{1}{2}})v_{j+\frac{1}{2}} + f(u_{j-\frac{1}{2}})v_{j-\frac{1}{2}} \]

Assuming unit cell

\[ v = x \]

\[ \frac{da_0}{dt} \int_{l_j} x \, dx + \frac{da_1}{dt} \int_{l_j} x^2 \, dx = \int_{l_j} f(a_0(t) + a_1(t)x) \, dx - f(u_{j+\frac{1}{2}})v_{j+\frac{1}{2}} + f(u_{j-\frac{1}{2}})v_{j-\frac{1}{2}} \]
Example: DG(1)

\[ v = x \]

\[ \frac{1}{2} \frac{da_0}{dt} + \frac{1}{3} \frac{da_1}{dt} = \int_{l_j} f(a_0 + a_1 x) \, dx - f(u_{j+\frac{1}{2}}) \]

Approximate with an internal quadrature

For 1D we can simply use:

Trapezoidal

\[ \approx \frac{1}{2} \left[ f(u_{j-\frac{1}{2}}) + f(u_{j+\frac{1}{2}}) \right] \]

\[ = \frac{1}{2} \left[ f(a_0) + f(a_0 + a_1) \right] \]

Simpsons

\[ \approx \frac{1}{6} \left[ f(u_{j-\frac{1}{2}}) + 4f \left( \frac{1}{2} \left[ u_{j-\frac{1}{2}} + u_{j+\frac{1}{2}} \right] \right) + f(u_{j+\frac{1}{2}}) \right] \]

\[ = \frac{1}{6} \left[ f(a_0) + 4f \left( \frac{1}{2} [2a_0 + a_1] \right) + f(a_0 + a_1) \right] \]
Example: DG(1) – Interfaces

- Now we have an explicit spatial discretization per cell
- Can view each cell as a separate entity that needs boundary data from its neighbors

- Next, find the fluxes across cell interfaces
Example: DG(1) – Numerical Flux

\[ v = x \]

\[ \frac{1}{2} \frac{da_0}{dt} + \frac{1}{3} \frac{da_1}{dt} = \frac{1}{2} [f(a_0^n) + f(a_0^n + a_1^n)] - f(u_{j+\frac{1}{2}}) \]

In general we approximate the flux with a numerical flux function

\[ f(u_{j+\frac{1}{2}}) \approx F_{j+\frac{1}{2}}(u_{j+\frac{1}{2}}^- , u_{j+\frac{1}{2}}^+) \]

2014-06-27
Choice of Numerical Flux Function

- Builds on extensive knowledge from development of numerical fluxes for FVM schemes
- Problem dependent
- Some examples:

Lax-Friedrichs:

\[ F^{LF}(u^-, u^+) = \frac{1}{2} \left( f(u^-) + f(u^+) - \alpha(u^+ - u^-) \right), \quad \alpha = \max_u |f'(u)| \]

Godunov:

\[ F^G(u^-, u^+) = \begin{cases} 
\min_{u^- \leq u \leq u^+} f(u), & \text{if } u^- < u^+ \\
\max_{u^+ \leq u \leq u^-} f(u), & \text{if } u^- \geq u^+ 
\end{cases} \]
Semi-Discrete Formulation

After inserting quadrature and numerical flux function, we are now left with two ODEs to solve in time:

\[ v = 1 \]

\[
\frac{da_0}{dt} + \frac{1}{2} \frac{da_1}{dt} = -F_{j+\frac{1}{2}}(u_j^{-}, u_j^{+})
\]

\[ v = x \]

\[
\frac{1}{2} \frac{da_0}{dt} + \frac{1}{3} \frac{da_1}{dt} = \frac{1}{2} [f(a_0^n) + f(a_0^n + a_1^n)] - F_{j+\frac{1}{2}}(u_j^{-}, u_j^{+})
\]
Explicit Discontinuous Galerkin

• Hyperbolic PDEs
  – Finite speed of propagation
  – Solution will propagate a certain distance over a period of time, $\Delta t$

• Locality of DG Method

• Use explicit time stepping/integration

• In the following we will use simple Euler time integration to evolve the solution in time
Time Integration

We then use an explicit solver in time, simple forward Euler in this case:

\[ v = 1 \]
\[ \frac{a_0^{n+1} - a_0^n}{\Delta t} + \frac{1}{2} \frac{a_1^{n+1} - a_1^n}{\Delta t} = -F_{j+\frac{1}{2}}(u_{j+\frac{1}{2}}, u_{j+\frac{1}{2}}) \]

\[ v = x \]
\[ \frac{1}{2} \frac{a_0^{n+1} - a_0^n}{\Delta t} + \frac{1}{3} \frac{a_1^{n+1} - a_1^n}{\Delta t} = \frac{1}{2} \left[ f(a_0^n) + f(a_0^n + a_1^n) \right] - F_{j+\frac{1}{2}}(u_{j+\frac{1}{2}}, u_{j+\frac{1}{2}}) \]

Compute \( R(u) \)
and solve linear system for \( a_0^{n+1} \) and \( a_1^{n+1} \)
Higher Order in Time

• For higher order in time, we can use, e.g., Runge-Kutta (RK)
• For the semi-discrete scheme:

\[ \frac{du}{dt} = R(u), \]

in which \( R(u) \) is the spatial DG discretization, the 3\textsuperscript{rd} order RK is:

\[
\begin{align*}
    u^{(1)} &= u^n + \Delta t R(u^n) \\
    u^{(2)} &= \frac{3}{4} u^n + \frac{1}{4} u^{(1)} + \frac{1}{4} \Delta t R(u^{(1)}) \\
    u^{n+1} &= \frac{1}{3} u^n + \frac{2}{3} u^{(2)} + \frac{2}{3} \Delta t R(u^{(2)})
\end{align*}
\]

CFL condition in 1D for up-to 2nd order spatial discretization: \(|c| \frac{\Delta t}{h} \leq \frac{1}{2p+1}\),

in which \(|c|\) is the wave speed for the given problem, and DG spatial discretization polynomials of degree \( p = k - 1 \) and RK method of order \( k \) are used
Higher Order in Space

1. **Modal**: Solutions are represented by sums of modal coefficients multiplied by a set of polynomials, e.g., \(u(x, t) = \sum_{i=1}^{N} u_i(t)P_i(x)\)
   - \(P_i\) is often orthogonal polynomials, e.g., Legendre (see left figure below)

2. **Nodal**: Cells are comprised of multiple nodes on which the solution is defined. Reconstruction of the cell is then based on fitting an interpolating polynomial, e.g., \(u(x, t) = \sum_{i=1}^{N} u_i(t)l_i(x)\)
   - \(l_i\) is a Lagrange polynomial (see right figure below)
Higher Order in Space

For the general case, in which we have $U \in \mathbb{R}$ and $P \in P^k$, we get a $(k + 1) \times (k + 1)$ system:

$$
\begin{bmatrix}
(1,1) & \cdots & (x^k, 1) \\
\vdots & \ddots & \vdots \\
(1, x^k) & \cdots & (x^k, x^k)
\end{bmatrix}
\begin{bmatrix}
a_0(t) \\
\vdots \\
a_k(t)
\end{bmatrix}
= 
\begin{bmatrix}
r_0 \\
\vdots \\
r_k
\end{bmatrix}
$$

$(p, q) = \int_{I_j} p(x)q(x)dx$

Remember: This system is local per cell

An orthogonal basis yields a diagonal matrix

Example of an orthogonal basis in 2D on triangles (generated with modepy)
Limiters

- For computing solutions with strong discontinuities, it is generally necessary to apply a limiter to prevent spurious numerical oscillations.
- This is typically done in a postprocessing step for DG schemes.
- Main idea: Replace the original DG-polynomial in "troubled cells" with another polynomial of the same order that is less oscillatory than the original.

![Original DG-polynomial and after applying limiter diagram]
Limiters

- The original cell average should be maintained for conservation
- Choice of limiter is highly problem dependent

- Builds upon knowledge from developing slope limiters for FVM
- For example: Minmod, generalized minmod, WENO

- See, e.g., *A simple weighted essentially nonoscillatory limiter for Runge-Kutta discontinuous Galerkin methods*, X. Zhong and C.-W. Shu
Extension to Higher Dimensions

We again start with a scalar conservation law, now in 2D

\[ u_t + f(u)_x + g(u)_y = 0 \]

Multiply with a test function \( v \) and integrate over some cell

\[ I_{jk} = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}] \times [y_{k-\frac{1}{2}}, y_{k+\frac{1}{2}}] \]

\[ \int_{I_{jk}} u_t v + f(u)_x v + g(u)_y v \, dx = 0 \]

Then integrate by parts

\[
\int_{I_{jk}} u_t v \, dV - \int_{I_{jk}} [f(u), g(u)] \cdot \nabla v \, dV + \int_{\partial I_{jk}} \mathbf{n} \cdot [f(u), g(u)] \, v \, dS = 0
\]

…and follow the derivation of the 1D scheme
Extension to Higher Dimensions

\[ \int_{I_{jk}} u_t v dx - \int_{I_{jk}} [f(u), g(u)] \cdot \nabla v \, dV + \int_{\partial I_{jk}} n \cdot [f(u), g(u)] \, v \, dS = 0 \]

Use a suitable quadrature rule (e.g., Gauss)

\[ \int_{I_{jk}} [f(u), g(u)] \cdot \nabla v \, dV = \sum_{m=0}^{1} \sum_{n=0}^{1} \omega_{mn} [f(u_{mn})(v_x)_{mn} + g(u_{mn})(v_y)_{mn}] \]
Parallel Computation

- DG *almost* perfectly parallel
- Divide into three computational *kernels*:

\[
\frac{du}{dt} = R(u) = N(u) + S(u^-, u^+)
\]

1. Time integration (solve ODEs)
2. Volume integral
3. Surface integral

- *Parallel Implementation of the Discontinuous Galerkin Method*, A. Baggag, H. Atkins, and D. Keyes
Parallel Computation

- Surface integrals depends only on neighboring cells, regardless of the order of the scheme
- For a structured 2D grid this leads to the following data dependency, assuming a block decomposition of the domain:

- Very similar to 1st order finite volume methods
Parallel Computation

- **FVM (FDM)**
  - Special treatment of boundaries necessary
  - Store one value per cell (per conserved variable)
  - High-order solution requires reconstruction \( \rightarrow \) larger stencil

- **DG**
  - Easy boundary conditions (choose a suitable boundary flux)
  - The higher DOF, the more values must be stored per cell
  - High-order solution in each cell
Parallel Computation

- New multi-core and many-core architectures can greatly speed up performance for parallel algorithms
- Since explicit DG is an embarrassingly parallel algorithm, it should run very efficiently on these architectures

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Multi-core CPUs</th>
<th>Xeon Phi</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain decomposition</td>
<td>MPI</td>
<td>MPI</td>
<td>NVIDIA CUDA</td>
</tr>
</tbody>
</table>
• National Center for Atmospheric Research’s (CISL’s Technology Development Division) evaluation of *accelerator* technology within existing weather and climate model codes

• DG_KERNEL benchmark performs a gradient operation from the Discontinuous Galerkin version of the High Order Methods Modeling Environment (HOMME)

• Specialized versions of the kernel were written for each target architecture (in cooperation with Intel for the Sandy Bridge CPU and the Xeon Phi)
PDE Frameworks Supporting RKDG

- **Diffpack**
  - Non-free license
  - Collection of C++ libraries and utility programs
  - Mature software & large community of users
  - Visualization: vtk, matlab, gnuplot, and more

- **Dune**
  - GPL 2-licensed (free)
  - C++ libraries with separation of data structures and algorithms by abstract interfaces
  - Mature software & large community of users
  - Visualization: vtk, GRAPE

- **Hedge**
  - MIT licensed (almost free)
  - Focus: Fast and easy to use
  - Python «front-end» and C++ «back-end»
  - Can run on CUDA-capable graphics card
  - Visualization: vtk, silo
Goals for Numerical Methods

1. Sufficient accuracy
   – High order and/or high grid resolution
   – Does the solution match analytical solutions? (Verification)
   – Does the solution replicate experimental data? (Validation)

2. Flexibility
   – Make solvers as generic as possible
   – Solve classes of problems using the same code

3. Robustness
   – Can we always expect a solution from our model?
   – Will it break down for, e.g., discontinuities or wet/dry-interfaces?
Goals for Numerical Methods

4. Efficiency & Scaling
   - How long does it take to compute a solution?
   - Convergence rate?
   - Consider computational platform: Desktop PC vs. MPI and supercomputer

5. Easy problem prototyping and code maintenance
   - Avoid ad-hoc solutions
   - Should be easy to set up a simulation (e.g., few lines of code)

   • Choice of method is often dictated by physical problem, domain complexity, required levels of accuracy, and available computational resources
Summary

• Hyperbolic problems
  – Arise in a multitude of different scientific areas
  – Typically form discontinuities – need to handle without smoothing the solution
  – Finite speed of propagation, which enables explicit time stepping

• Discontinuous Galerkin
  – Locality of FVM combined with variable DOF per element
  – General and flexible framework for solving large classes of PDEs
  – Conceptually no difference between 1D, 2D or N-D
  – Support for locally adaptive numerical solutions ($hp$-adaptivity), and meshes can be both non-conforming and unstructured
  – The method is highly suitable for parallel computations – only depends on nearest neighbors in grid
Further Reading

• Book by Hesthaven and Warburton

• PhD-course slides by Hesthaven (inspired some of my slides)
Further Reading

• Essay for a wider audience:
  – COCKBURN, B: *Discontinuous Galerkin Methods*

• Runge-Kutta Discontinuous Galerkin Method:

• Development and state-of-the-art pre-1999:

• Updated review: