An Introduction to Discontinuous Galerkin Methods

Module 3B: To Higher-Orders - Discrete System

J. Bevan

Department of Mechanical Engineering, Grad Student
University of Massachusetts at Lowell

Module 3B: To Higher-Orders - Discrete System

- Numerical Quadrature (Gauss)
- Hermite Interpolation (and quadrature)
- Truncation error/exact quadrature
- GL Lagrange Orthogonality
- Local Mapping Function (Jacobian)
- Mass Matrix- Diagonalization
- Log differentiation
- Stiffness Integral
- Numerical Flux (Extrapolated)
- Assembly of System
- RK4 Time discretization
- Investigate p-Convergence

Numerical Quadrature (Gauss)

- We now have a method for generating a robust arbitrary order solution approximation, but unlike before it isn't practical to analytically pre-calculate all the integrals.
- We can use a numerical quadrature technique to do this instead, able to integrate arbitrary functions
- If we call the interpolation of a function Inf then we assume that for a sufficiently accurate interpolation, we can use the interpolation of the function wherever we could use the function itself before. This is in general M-1 order accurate.

$$f \approx Inf = \sum_{i=0}^{M} f(x_i) L_i(x) \tag{1}$$

$$\int f \approx \int Inf = \int \sum_{i=0}^{M} f(x_i) L_i(x) = \sum_{i=0}^{M} f(x_i) \int L_i(x) = \sum_{i=0}^{M} f(x_i) w_i \tag{2}$$

Hermite Interpolation (and quadrature)

- Recall: Hermite interpolation includes derivatives of interpolated function as well
- Consider a Hermite interpolation polynomial that includes first derivatives as well, the interpolation would be $2M-1$ accurate. The quadrature using this polynomial would look like:

$$\int f\,dx \approx \sum_{i=0}^{M} f(x_i)w_i + \sum_{i=0}^{M}\left[f'(x_i)\int (x-x_i)L_i^2(x)\,dx\right] \quad (3)$$

- It turns out if we choose our quadrature/interpolation points to be the Legendre roots, the integral for the second term is zero. Thus no first derivative terms are needed for the Hermite quadrature, even though it is $2M-1$ order accurate.

Truncation error/exact quadrature

- It is important to consider error sources from the approximation to the solution and integrals, these can affect convergence
- Three main error sources in quadrature: aliasing, truncation, and inexact quadrature
- Aliasing occurs if the function is not sampled frequently enough, it is assumed that the sol'n is sufficiently smooth and the discretization suitably fine to avoid this in most cases
- Truncation is unavoidable except where the exact function is of equal or lesser order than the interpolation/quadrature. Higher order terms present in the exact function are left off.
- Inexact quadrature occurs when the total polynomial order of the product of the interpolated functions undergoing quadrature exceeds the exactness of the quadrature. For Gauss-Legendre quadrature this isn't a problem for one and even two functions in the integrand. Each function is of order M-1 and the quadrature is exact for 2M-1, so the quadrature is able to exactly integrate the interpolation

GL Lagrange Orthogonality

- A final useful property of the Lagrange basis with Legendre interpolation points is orthogonality
- The product of two $M-1$ order Lagrange bases can be rearranged to be a Legendre poly of order M and a remainder polynomial of order $M-2$
- The remainder polynomial can be expressed as a linear combination of Legendre polys all of order $< M$, all are orthogonal to the order M Legendre, so

$$\int_{-1}^{1} L_i(x) L_j(x)\, dx = \delta_{ij} w_i \tag{4}$$

Local Mapping Function (Jacobian)

- The domain of orthogonality for Legendre polynomials (and by extension GL Lagrange) is $[-1, 1]$
- Elements may be arbitrary sizes though, we'd like to be able to transform $x \in [x_L, x_R] \rightarrow X \in [-1, 1]$ by means of a mapping $x = g(X)$ and it's inverse $X = G(x)$

$$x = g(X) = \frac{X+1}{2}\Delta x + x_L \quad , \quad X = G(x) = \frac{2(x - x_L)}{\Delta x} + 1 \tag{5}$$

- Applying a change of variables for the mapping

$$\int_{x_L}^{x_R} L_i(x) L_j(x)\, dx \rightarrow \int_{-1}^{1} L_i(g(X)) L_j(g(X)) g'\, dX \tag{6}$$

- $J = g' = \frac{\Delta x}{2}$ is called the determinant of the Jacobian matrix

Mass Matrix- Diagonalization

- We have done what seems like quite a bit of tangential work, but it now pays off

$$\sum_{i=0}^{M} \frac{d\widetilde{q}_i}{dt} \int_k \psi_i(x)\phi_j(x)\, dx = \sum_{i=0}^{M} \frac{d\widetilde{q}_i}{dt} \int_I L_i(X)L_j(X)\frac{\Delta x}{2}\, dX \tag{7}$$

$$= \frac{\Delta x}{2}\sum_{i=0}^{M} \frac{dq_i}{dt}\delta_{ij}w_i = \frac{\Delta x}{2}q_j' w_j \quad for\, all\, j \tag{8}$$

- We've reduced the full mass matrix into a diagonal mass matrix with all other terms zero, compared to Module 2 solver case the mass matrix is trivially invertible. We have: $\frac{\Delta x}{2}\mathbf{q}'\,\mathbf{M}$ where $\mathbf{M}_{jj} = w_j$

Log differentiation

- We can get a closed form expression for $L_j'(x)$ in the stiffness term by using logarithmic differentiation
- The main idea is that in general $f'/f = ln(f)$ so applying this to our Lagrange basis

$$L_j'(x) = L_j \sum_{r=0, r\neq j}^{N} \frac{1}{x - x_r} \tag{9}$$

- using our previous function code for "*Lag(x)*" we can get a general expression *dLag= @(x,nv) Lag(x,nv).*sum(1./bsxfun(@minus,x,nn(nv,:,:)),3)*
- We need a more involved method for evaluation at the interp points (see dLagrange.m)

Stiffness Integral

- We now have a routine for calculating L'_j, and suitable quadrature; we can evaluate the stiffness integral

$$\int_k \sum_{i=0}^{M} \left[c\widetilde{q}_i \psi_i(x) \right] \phi'_j(x)\, dx = \sum_{i=0}^{M} c\widetilde{q}_i \int_I L_i(X) L'_j(X)\, dX \tag{10}$$

- No Jacobian term from the mapping. The derivative in the integrand produces a complementary $1/J$ that cancels due to the change of variables.
- No tricks to be had for reducing the stiffness matrix, it is a full matrix. We have: $c\mathbf{K}_{ji}\,\widetilde{\mathbf{q}}$

Numerical Flux (Extrapolated)

- One downside of using Gauss-Legendre points is there are no points on the boundary
- It is easy to calculate the boundary solution values from the solution interpolation at the left end (same idea for the right)

$$\widetilde{q}(x_L) = \sum_{i=0}^{M} \widetilde{q}_i L_i(x_L) \tag{11}$$

- Call $L_i(x_L) = L_{iL}$, the vector notation is then $\widetilde{q}_L = \mathbf{L}_{iL}^T \widetilde{\mathbf{q}}$
- So that our numerical flux vector is $\hat{\mathbf{f}} = c(\overset{k}{q}_R \mathbf{L}_{jR} - \overset{k-1}{q}_R \mathbf{L}_{jL})$
- Lobatto alternative gives boundary points, but would make the mass matrix a full matrix. Inversion is likely more expensive than interpolation

Assembly of System

- We can now combine the mass, stiffness, and numerical flux terms
$$\frac{\Delta x}{2}\mathbf{q}'\,\mathbf{M} + \hat{\mathbf{f}} - c\mathbf{K}_{ji}\,\widetilde{\mathbf{q}} = 0 \tag{12}$$
- solving for q'
$$\widetilde{\mathbf{q}}' = \frac{2}{\Delta x}(c\mathbf{K}_{ji}\,\widetilde{\mathbf{q}} - \hat{\mathbf{f}})\mathbf{M}^{-1} \tag{13}$$
- it is also possible to represent it componentwise easily thanks to the diagonal mass matrix
$$\widetilde{q}_j' = \frac{2}{w_j\Delta x}(c\mathbf{K}_{j\text{-}}\widetilde{\mathbf{q}} - \hat{f}_j) \tag{14}$$

RK4 Time discretization

- Compared to the simple linear DG solver, we'd like to use a higher order time discretization, Runge-Kutta 4th order: RK4. We can express as a function: $\widetilde{\mathbf{q}}'(\widetilde{\mathbf{q}})$ from our discrete system. RK4 consists of 4 trial steps:
  $k_1 = \widetilde{\mathbf{q}}'(\widetilde{\mathbf{q}})\,\Delta t$
  $k_2 = \widetilde{\mathbf{q}}'(\widetilde{\mathbf{q}} + \frac{k_1}{2})\,\Delta t$
  $k_3 = \widetilde{\mathbf{q}}'(\widetilde{\mathbf{q}} + \frac{k_2}{2})\,\Delta t$
  $k_4 = \widetilde{\mathbf{q}}'(\widetilde{\mathbf{q}} + k_3)\,\Delta t$
  $\mathbf{q}(t+1) = \mathbf{q} + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$

Investigate p-Convergence

- How does the p-convergence rate compare with h-convergence?
- Does the smoothness of the initial sol'n seem to effect the rate of convergence? (e.g. sin(x) vs gaussian curve)
- Does h or p refinement seem to be more efficient?