Discontinuous Galerkin implementation in foam-extend

Report-1
September 2018

Gregor Cvijetić

University of Zagreb, Croatia, gregor.cvijetic@fsb.hr

Contents

Summary

In this report the implementation of discontinuous Galerkin (DG) method in foam-extend is presented. As reports will be delivered on a monthly basis, of which this is the first one, the fundamental programming and mathematical basis, as well as the core code structure are presented here. Each following report will continue from where the previous report left off, without repeating parts that are not in the focus of the current work. Therefore, all of the reports should be considered as a whole, rather than each report as an independent document.

The report contains:

1. The mathematical basis for discontinuous Galerkin method,
2. The mathematical model for Poisson equation,
3. Discussion on the code structure,
4. Details of the implementation and numerical procedures.

In the section Mathematical model an overview of the implemented mathematical approach will be given. Furthermore, the Gauss integration and choice of modal basis will be briefly covered. In the section Code structure the code structure will be presented. As the code structure adheres to the foam-extend standard, only the overview of the complete code will be given, while parts of the code that are specific to DG will be highlighted. Section Implementation details presents a walkthrough of important implementation parts needed for code understanding and getting familiar with how the solver works.

1 Mathematical model

In this section the mathematical model of the discontinuous Galerkin method will be presented. No theory will be covered as it is accessible in a number of sources **??**, therefore only equations and relevant parts of theory will be covered.

1.1 Modal basis

As opposed to available alternative (nodal basis), a modal basis approach has been chosen in this work. This means that referent cell value is calculated as a superposition of modes (polynomials), whereas only the polynomial coefficients (weights on modal values) φ_i are stored:

$$u(x) = \sum_{i=1}^{N} \varphi_i P_i(x)\,, \tag{1}$$

where N represents number of modes used, P_i is a polynomial of mode i evaluated at coordinate x and φ_i is a polynomial coefficient. The equation (1) is valid cell-wise, but at this point refers only to a referent cell, meaning that x refers to a local coordinate, $x \in \{-1, 1\}$.

The choice of polynomials is arbitrary, while due to benefitial properties **??** the Legendre polynomials are used here:

$$\begin{aligned} P_0 &= 1 \\ P_1 &= x \\ P_2 &= \frac{1}{2}(3x^2 - 1) \\ P_3 &= \frac{1}{2}(5x^2 - 3x) \\ P_4 &= \frac{1}{8}(35x^4 - 30X^2 + 3) \\ &... \end{aligned} \tag{2}$$

or using the recursive formula:

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x) \tag{3}$$

with

$$P_0(x) = 1 \text{ and } P_1(x) = x\,. \tag{4}$$

For the initial work only modes up to second order are implemented and the implementation for 1-D problems is performed.

1.2 Gaussian quadrature integration

Gaussian quadrature rule refers to approximation of the definite integral of a function, formulated as a weighted sum of function values at specified points within the domain of integration:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)\,, \tag{5}$$

Points x_i at which the function is evaluated are called Gauss nodes. The accuracy of the approximation depends on the number of used points, whereas in case of Legendre polynomials of the order $2n-1$, n Gauss nodes are needed to obtain the exact value of the integral. Weighting factors w_i are calculated using the relation:

$$w_i = \frac{2}{(1 - x_i^2)[P_n'(x_i)]^2} \tag{6}$$

while coordinate x_i is the i-th root of the polynomial P_n. The quadrature rule is used for evaluating the volume and surface integrals throughout the code.

TREBAM LI PROVJERITI DA LI SE INTEGRAL RACUNA SAMO U PAR TOCAKA ILI BI SVAKI POLINOM TREBAO U SVOJIM TOCKAMA

1.3 Poisson equation

First task of the implementation is to solve the Poisson equation. The discretisation is performed using the Symmetric Interior Penalty (SIP) method, with the aid of the mean value operator $\{\cdot\}$ and jump operator $[\![\cdot]\!]$ on the cell edges, due to the discontinuous nature of cell-wise polynomials. Following [**?**] the discretized form of Poisson equation multiplied by a test function states:

$$\int_\Omega \nabla_\bullet(\nabla u)v dV = \int_\Omega \nabla u \cdot \nabla v dV - \oint_\Gamma \{\nabla u\} \cdot n_\Gamma [\![v]\!] dA - \oint_\Gamma \{\nabla v\} \cdot n_\Gamma [\![u]\!] dA + \oint_\Gamma \eta [\![u]\!][\![v]\!] dA \quad (7)$$

where on the right hand side of the equation terms from left to right are: volume term, consistency term, symmetry term and penalty term. Jump and mean value operators are:

$$\{c_h\} = \frac{c_h^- + c_h^+}{2} \quad (8)$$

$$[\![c_h]\!] = c_h^- - c_h^+ \quad (9)$$

2 Code structure

2.1 Source code

2.2 Unit tests

2.3 Solvers

3 Implementation details

3.1 Primitives

3.2 Mesh

3.3 Fields

3.4 Matrix

3.5 Operators

3.6 Boundary Conditions

Napisati da ce biti provedena usporedba s Bosss code i Kummer's polynomials.

Osvrnuti se na numerical flux