Notes on weight-adjusted discontinuous Galerkin methods with Bernstein-Bezier basis functions

1 Projection

For non-constant coefficients, DG requires being able to deal with polynomial multiplication and projection onto lower-dimensional subspaces. Multiplying polynomials together may be done using a discrete convolution and polynomial multiplication (J. Sanchez-Reyes 2003). The projection operator may be derived by noting that degree elevation operators are diagonal when transformed to a modal basis.

1.1 Polynomial multiplication

Rescaling by binomial coefficients results in the unscaled Bernstein basis. Polynomial multiplication is then equivalent to discrete convolution of the scaled binomial coefficients.

Quadrature-free strategy for nonlinear volume terms: polynomial multiplication + projection.

1. Polynomial multiplication of two BB basis functions representable as coefficient scaling, N_p scalar multiplications and storage of N_p coeffs, and another coefficient scaling.
2. To reduce local memory costs, process coeffs for fg over one or more $(d-1)$ dimensional layers.
3. Store ids and load

1.2 Projection operator

Waldron showed that this projection operator has a form

$$\boldsymbol{M}_N^{-1}\boldsymbol{M}_{N,M} = \sum_{j=0}^{N} c_j E_{N-j}^{N}\left(E_{N-j}^{M}\right)^T.$$

The constants c_j may be computed through the solution of an $(N+1)\times(N+1)$ matrix system, using the fact that upon transformation to a modal basis, E_{N-j}^N is a diagonal matrix of ones and zeros, while E_{N-j}^M is a diagonal matrix with entries

$$\frac{\lambda_i^{N-j}}{\lambda_i^M}, \qquad i = 0,\ldots,N.$$

This may be factored into an application of E_N^M, then an application of

$$\begin{aligned}\sum_{j=0}^{N} c_j E_{N-j}^{N}\left(E_{N-j}^{N}\right)^T &= c_0\boldsymbol{I} + c_1 E_{N-1}^N\left(E_{N-1}^N\right)^T + c_2 E_{N-1}^N E_{N-2}^{N-1}\left(E_{N-2}^{N-1}\right)^T\left(E_{N-1}^N\right)^T + \ldots\\ &= c_0\boldsymbol{I} + c_1 E_{N-1}^N\left(I + \frac{c_2}{c_1}E_{N-2}^{N-1}\left(I+\ldots\right)\left(E_{N-2}^{N-1}\right)^T\right)\left(E_{N-1}^N\right)^T.\end{aligned}$$

This may be applied in two sweeps of length N, using in-place updates to memory. On GPUs, this will unfortunately still require synchronizations between each application.

The cost of applying $\left(E_N^M\right)^T$ is the application of $(M - N)$ sparse degree elevation operations, each of which is $O(M^d)$ cost. Assuming $M \approx N$ (it is reasonable to match the order of the data with the order of approximation), this gives $O(N^{d+1})$ cost.

When applying the projection operator, since each operation is $O(N^3)$ and we apply $O(N)$ total operations, we have an $O(N^{d+1})$ overall cost.