Development of nodal basis sets on simplex elements

Jack Coughlin

July 28, 2023

Contents

Introduction

This writeup describes the notation and exact definitions of the basis element matrices used in WARPXM. Readers should be familiar with the nodal DG formulation, in particular the formulation given in Hesthaven and Warburton's Nodal Discontinuous Galerkin Methods (Springer 2008). We make several departures from their derivation. Chiefly among them, we assume the existence of exact quadrature rules for the triangle (and tetrahedron, TODO). This lets us express basis element matrices such as the mass matrix in terms of integration, rather than identities based on the Vandermonde matrix. It is hoped that this first-principles approach is more flexible than the Vandermonde matrix approach.

1 Notation and dimension-independent formulation

Despite attempting to depart from the dependency on the Vandermonde matrix and work out the basis matrices from first principles, we will find ourselves using more Vandermonde matrices than ever before. It's worth recollecting what a Vandermonde matrix is and how it may be used. We'll denote all Vandermonde matrices by

$$\mathcal{V}^{\psi,\boldsymbol{y}}, \tag{1}$$

where $\boldsymbol{y}$ is a collection of points and ψ is a family of polynomials. Then,

$$\mathcal{V}^{\psi,\boldsymbol{y}}_{ij} = \psi_j(\boldsymbol{y}_i).$$

That is, the columns are different polynomials, and the rows are different points.

Let's denote the element we want to discretize by Ω, and let N be the maximum (total) polynomial degree.

The first step is to define an orthogonal polynomial basis, which we denote $\phi_j(\boldsymbol{x})$. The polynomials have the defining relation

$$\int_\Omega \phi_i(\boldsymbol{x})\phi_j(\boldsymbol{x})\,d\boldsymbol{x} = \delta_{ij}\alpha_i, \tag{2}$$

where δ_{ij} is the Kronecker delta and α_i is a normalization constant.

Now introduce a vector of collocation nodes, $\boldsymbol{x}_i$. These should be chosen so that the condition number of the generalized Vandermonde matrix (defined below) is minimized. In one dimension, the Legendre-Gauss-Lobatto points are a good choice. In higher dimensions, more complex procedures are required.

The generalized Vandermonde matrix for the family ϕ is

$$\mathcal{V}^{\phi,\boldsymbol{x}}_{ij} = \phi_j(\boldsymbol{x}_i). \tag{3}$$

Now, there is also a Lagrange interpolating basis ℓ_i, which satisfies

$$\mathcal{V}^{\ell,\boldsymbol{x}}_{ij} = \ell_j(\boldsymbol{x}_i) = \delta_{ij}. \tag{4}$$

In one dimension these have an explicit formula, but in higher dimensions they do not. To evaluate them, then, we observe that the interpolating basis has a modal expansion,

$$\ell_j(\boldsymbol{y}) = \sum_k \phi_k(\boldsymbol{y})\hat{\ell}_{kj}, \tag{5}$$

for all $\boldsymbol{y} \in \Omega$. But this is true for the generalized Vandermonde matrix as well:

$$\delta_{ij} = \ell_j(\boldsymbol{x}_i) = \sum_k \phi_k(\boldsymbol{x}_i)\hat{\ell_{kj}} = \sum_k \mathcal{V}^{\phi,\boldsymbol{x}}_{ik}\hat{\ell_{kj}} = \left[\mathcal{V}^{\phi,\boldsymbol{x}}\hat{\ell}\right]_{ij} = \delta_{ij}. \tag{6}$$

So we can invert the Vandermonde matrix to find the ϕ expansion of the Lagrange interpolating polynomials:

$$\hat{\ell}_{kj} = [(\mathcal{V}^{\phi,\boldsymbol{x}})^{-1}]_{kj}. \tag{7}$$

To reiterate, to evaluate the Lagrange interpolating polynomial ℓ_j at an arbitrary point $\boldsymbol{y}_i$, we use the formula

$$\mathcal{V}^{\ell,\boldsymbol{y}}_{ij} = \ell_j(\boldsymbol{y}_i) = \sum_k \phi_k(\boldsymbol{y}_i)[(\mathcal{V}^{\phi,\boldsymbol{x}})^{-1}]_{kj} = \left[\mathcal{V}^{\phi,\boldsymbol{y}}(\mathcal{V}^{\phi,\boldsymbol{x}})^{-1}\right]_{ij}. \tag{8}$$

We will also need to be able to evaluate derivatives of the Lagrange interpolating polynomials. This can be done via Vandermonde matrices for the derivatives of the orthogonal basis, as follows:

$$\mathcal{V}^{\partial_r\ell,\boldsymbol{y}}_{ij} = \frac{\partial \ell_j}{\partial r}(\boldsymbol{y}_i) = \sum_k \frac{\partial \phi_k}{\partial r}(\boldsymbol{y}_i)\left[(\mathcal{V}^{\phi,\boldsymbol{x}})^{-1}\right]_{kj} = \left[\mathcal{V}^{\partial_r\phi,\boldsymbol{y}}(\mathcal{V}^{\phi,\boldsymbol{x}})^{-1}\right]_{ij}. \tag{9}$$

1.1 Exact quadrature

For each element type, we can construct quadrature rules which integrate multivariate polynomials of up to total degree $2N$ exactly. This can be done by projecting tensor-product quadrature rules onto the simplex domain, in cases where an optimal Gaussian quadrature rule is not available.

A quadrature rule is defined by a vector of abscissas (nodes) $\boldsymbol{q}_i$ and a vector of weights, w_i. For an arbitrary polynomial h of degree up to $2N$,

$$\frac{1}{|\Omega|}\int_\Omega h(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_i w_i h(\boldsymbol{q}_i), \quad \sum_i w_i = 1, \tag{10}$$

where $|\Omega|$ is the multidimensional volume of the element Ω. For a typical basis matrix, whose elements are the L^2 inner product of a family of functions, we can write it in terms of a matrix product involving Vandermonde matrices and a diagonal matrix of the quadrature weights. For example, letting f and g be arbitrary polynomials of degree at most N,

$$\int_\Omega f_i(\boldsymbol{x})g_j(\boldsymbol{x})\,d\boldsymbol{x} = \sum_k w_k f_i(\boldsymbol{q}_k)g_j(\boldsymbol{q}_k) = \sum_k \mathcal{V}^{f,\boldsymbol{q}}_{ki} w_k \mathcal{V}^{g,\boldsymbol{q}}_{kj} = (\mathcal{V}^{f,\boldsymbol{q}})^T D(w)\mathcal{V}^{g,\boldsymbol{q}}, \tag{11}$$

where $D(w)$ is the diagonal matrix with entries $D_{ii} = w_i$.

1.2 Basis matrices

We are now in a position to evaluate the basis matrices, whose entries are integrals over the element Ω. In this section, assume that the nodes $\boldsymbol{q}$ and weights w correspond to an exact quadrature rule.

Remark: serialization of basis matrices

The matrices described below are flattened and written into `.txt` files in the WARPXM source tree. They are written out in a row-major order.

Mass matrix

$$\mathcal{M}^{\ell}_{ij} = \int_{\Omega} \ell_i(\boldsymbol{x})\ell_j(\boldsymbol{x})\,d\boldsymbol{x} = (\mathcal{V}^{\ell,\boldsymbol{q}})^T D(w)\mathcal{V}^{\ell,\boldsymbol{q}}. \tag{12}$$

This matrix and its inverse are serialized (flattened) and used in WARPXM:

$$\texttt{MASS} = \mathcal{M}, \quad \texttt{INV\_MASS} = \mathcal{M}^{-1}. \tag{13}$$

It is computed by the `lagrange_mass_matrix` function in the Python scripts.

Advection matrix

$$\mathcal{A}^{r,\ell}_{ij} = \int_{\Omega} \ell_i(\boldsymbol{x})\frac{\partial \ell_j}{\partial r}(\boldsymbol{x})\,d\boldsymbol{x} = (\mathcal{V}^{\ell,\boldsymbol{q}})^T D(w)\mathcal{V}^{\partial_r\ell,\boldsymbol{q}}. \tag{14}$$

There is one advection matrix per dimension. The matrix that gets serialized and used in WARPXM is

$$\texttt{UPSILON\_2} = \mathcal{M}^{-1}\mathcal{A}^T. \tag{15}$$

In the code this is referred to as the `internalFluxArray` or `internal_flux_array`.
Another matrix that gets serialized and used in WARPXM is called the `derivativeBasisArray`:

$$\texttt{Dr\_Basis\_Array} = \mathcal{M}^{-1}\mathcal{A}. \tag{16}$$

These are calculated by the python function `derivative_arrays`.

Lift matrices

For a given face F, we need to be able to compute face integrals of the numerical flux. This is done via a face quadrature rule, whose nodes and weights we denote by $\boldsymbol{q}^f, w^f$.

$$\mathcal{E}_{ij} = \int_F \ell_i(\boldsymbol{x})\ell(\boldsymbol{x})\,d\boldsymbol{x} = (\mathcal{V}^{\ell,\boldsymbol{q}_f})^T D(w_f)\mathcal{V}^{\ell_F,\boldsymbol{q}_f}. \tag{17}$$

Here, the Vandermonde matrix $\mathcal{V}^{\ell_F,\boldsymbol{q}_f}$ indicates the evaluation of those Lagrange polynomials whose collocation nodes lie on the face F, at the quadrature points $\boldsymbol{q}_f$ on that face. There is one lift matrix per face, and the matrix that gets serialized and used in WARPXM is actually

$$\texttt{UPSILON\_1} = \mathcal{M}^{-1}\mathcal{E}. \tag{18}$$

This family of matrices is computed by the `lift_matrices` Python function.

Gaussian Quadrature source term mass matrix

When evaluating a source term via Gaussian quadrature, one uses an integral of the form

$$\int_\Omega \ell_i(\boldsymbol{x}) S(\boldsymbol{x})\, d\boldsymbol{x} \approx \sum_k \ell_i(\boldsymbol{q}_k) w_k S(\boldsymbol{q}_k). \tag{19}$$

Then, when forming the right hand side, this sum will be multiplied by the inverse mass matrix, in a term of the form

$$\left[(\mathcal{M}^\ell)^{-1}\right] \left[D(w)\mathcal{V}^{\ell,\boldsymbol{q}}\right]^T \boldsymbol{S}, \tag{20}$$

where $[\boldsymbol{S}]_k = S(\boldsymbol{q}_k)$. This matrix is serialized and passed to WARPXM as

$$\texttt{UPSILON\_3} = \mathcal{M}^{-1} D(w) \mathcal{V}^{\ell,\boldsymbol{q}}. \tag{21}$$

This is used in tandem with the Vandermonde matrix for the Lagrange polynomials evaluated at the quadrature nodes:

$$\texttt{LQUAD} = \mathcal{V}^{\ell,\boldsymbol{q}}. \tag{22}$$

Element averages

Taking the average of a nodal function over the element may be accomplished via the dot product with the vector

$$\boldsymbol{b}^\ell_j = D(w)\mathcal{V}^{\ell,\boldsymbol{q}}_{ij} \tag{23}$$

Conversion to monomials

To convert *from* a monomial representation to a nodal expansion at the collocation points, we should use the Vandermonde matrix of monomials M evaluated at the collocation points. To see this, consider the monomial expansion of u with coefficients a_k. By definition we should have

$$u(\boldsymbol{x}_i) = \sum_k a_k \boldsymbol{x}_i^k, \tag{24}$$

or in matrix notation,

$$u(\boldsymbol{x}_i) = (\mathcal{V}^{M,\boldsymbol{x}}_{ik})\boldsymbol{a}_k. \tag{25}$$

Thus, converting from the nodal values $u(\boldsymbol{x}_i)$ to the monomial coefficients is just an inversion of $\mathcal{V}^{M,\boldsymbol{x}}$. This matrix is serialized and used in WARPXM as

$$\texttt{CONVERT\_TO\_MONOMIAL} = \left(\mathcal{V}^{M,\boldsymbol{x}}\right)^{-1}. \tag{26}$$

Evaluation at positivity-preserving quadrature nodes

The `positivity_preserving_dg.pdf` writeup, which can be found in this same directory, describes the use of quadrature rules with all positive weights to preserve positivity of a high-order DG solution. The upshot of that positivity-preserving scheme is that for a given reference element, there are some number of "extra" nodes, at which we must be able to evaluate the solution value. These are nodes in excess of the element's face nodes and its interior LGL nodes.

The number of such extra nodes is given in the `NUM_INTERIOR_POSITIVITY_PRESERVING_QUAD_NODES` property of the basis array file. If this is nonzero, there is another line in the file, `EVALUATE_AT_INTERIOR_POSITIVITY_PRESERVING_QUAD_NODES`, which is the matrix mapping LGL nodal values to values at the extra positivity nodes $\boldsymbol{p}$:

$$\texttt{EVALUATE\_AT\_INTERIOR\_POSITIVITY\_PRESERVING\_QUAD\_NODES} = \mathcal{V}^{\ell,\boldsymbol{p}}, \quad (27)$$

2 Line element

For a reference line element, we use the interval $[0, 1]$ for consistency with the triangle and tetrahedral reference elements. Exact quadratures are achieved with the Gauss-Legendre quadrature rule of a given order, which is also optimal. For collocation nodes we use the Legendre-Gauss-Lobatto points. Finally, all weights of the LGL quadrature are positive, so there is no need for extra interior positivity nodes.

3 Triangle

3.1 Face quadrature

For exact quadrature over the faces, we use the Legendre-Gauss-Lobatto quadrature rule with number of nodes equal to the number of face nodes.

3.2 Exact volume quadrature rule

This subsection describes the construction of an exact quadrature rule on the triangle. This rule is *not* used in WARPXM at runtime. It is only used to compute the basis arrays from their definitions in terms of inner products of polynomials. For this purpose, in a simulation whose unknowns may be polynomials of up to total degree k, we require a quadrature rule which is exact for polynomials of total degree $2k$.

To construct an exact quadrature rule for polynomials of arbitrary degree on the unit ($[0, 1]$) triangle, we follow a similar procedure to the one outlined in [1]. We begin with a tensor-product quadrature rule on the unit square. Our aim is not to obtain an optimal or even particularly efficient quadrature rule on the triangle, just to get a rule that will be exact.

We begin by defining a mapping from the square with coordinates $(u, v) \in [-1, 1]^2$,

$$g(u, v) = \frac{1+v}{2} V_1 + \frac{1+u}{2}\frac{1-v}{2} V_2 + \frac{1-u}{2}\frac{1-v}{2} V_3, \tag{28}$$

where

$$V_1 = (0, 1), \quad V_2 = (1, 0), \quad V_3 = (0, 0). \tag{29}$$

The determinant of the Jacobian of g is

$$|J| = |\frac{\partial g(u, v)}{\partial (u, v)}| = \frac{1-v}{2}. \tag{30}$$

Now, an integral over the $[0, 1]$ simplex Ω is given by

$$\int_\Omega p(x, y)\, dxdy = \int_0^1 \int_0^1 p(g(u, v))|J|\, dudv. \tag{31}$$

Suppose that $p(x, y)$ is a polynomial of total degree $2k$; then the integrand $\hat{p}(u, v) = p(g(u, v))|J|$ is a polynomial of up to degree $2k$ in u, and degree $2k+1$ in v. The integral over the (u, v) square may therefore be computed exactly via a tensor product quadrature rule, which must be exact for polynomials of degree up to $2k$ in u, and degree up to $2k + 1$ in v. One choice is the tensor product of LGL quadrature rules in u and v. The N point LGL quadrature rule is exact for polynomials of degree up to $2N - 3$, so we need N such that $2N - 3 \geq 2k + 1$, or $N \geq k + 2 = N_p + 1$.

This choice of quadrature rule is computed by the Python function `triangle_exact_quadrature_rule`.

3.3 Orthonormal basis

Hesthaven and Warburton suggest the following basis, which we modify slightly for the $[0, 1]$ triangle:

$$\phi_{i,j} = \sqrt{2} P_i^{(0,0)}(a) P_j^{(2i+1,0)}(b)(1 - b)^i, \tag{32}$$

where

$$a = 2\frac{r}{1-s} - 1, \quad b = 2s - 1, \tag{33}$$

with $(r, s) \in \Omega$.

3.4 Extra positivity-preserving quadrature nodes

For details on how the extra positivity-preserving quadrature nodes (positivity nodes) are used, see the positivity_preserving_dg.pdf writeup. This section describes how the nodes are chosen, following a modification of the procedure suggested by Zhang et al. (section 3.2 of [1]).

N_p	k	N_{LGL} in u	N_{LGL} in v
2	1	2	3
3	2	3	3
4	3	4	4

Table 1: Required numbers of quadrature nodes for the exact integration of $\hat{p}(u, v) = p(g(u, v))|J|$, assuming $p(x, y)$ has total degree k.

We first note that the procedure described above in Section 3.2 results in a quadrature rule whose weights are all positive (since it is based on a tensor product of quadrature rules with positive weights). Where possible, we use the same quadrature rule in u and v as we use for the face integrals, namely the rule with $N_{LGL} = N_p$. Only when $N_p = 2$ does this not suffice; in that case, the degree of $\hat{p}$ in v may be as high as 3, requiring a 3-point LGL quadrature rule. To obtain the desired quadrature rule with all positive weights, we average the three rules resulting from each of the three projections of the square onto the triangle, described above. This results in a rule which shares some nodes with the element's face nodes, as well as some extra nodes. The process is illustrated in Figure 1

4 Tetrahedra

TODO

References

[1] Xiangxiong Zhang, Yinhua Xia, Chi-Wang Shu. "Maximum-Principle-Satisfying and Positivity-Preserving High Order Discontinuous Galerkin Schemes for Conservation Laws on Triangular Meshes". Journal of Scientific Computation, (2012).

[2] Jan Hesthaven and Tim Warburton. "Nodal Discontinuous Galerkin Methods". Springer, 2008.

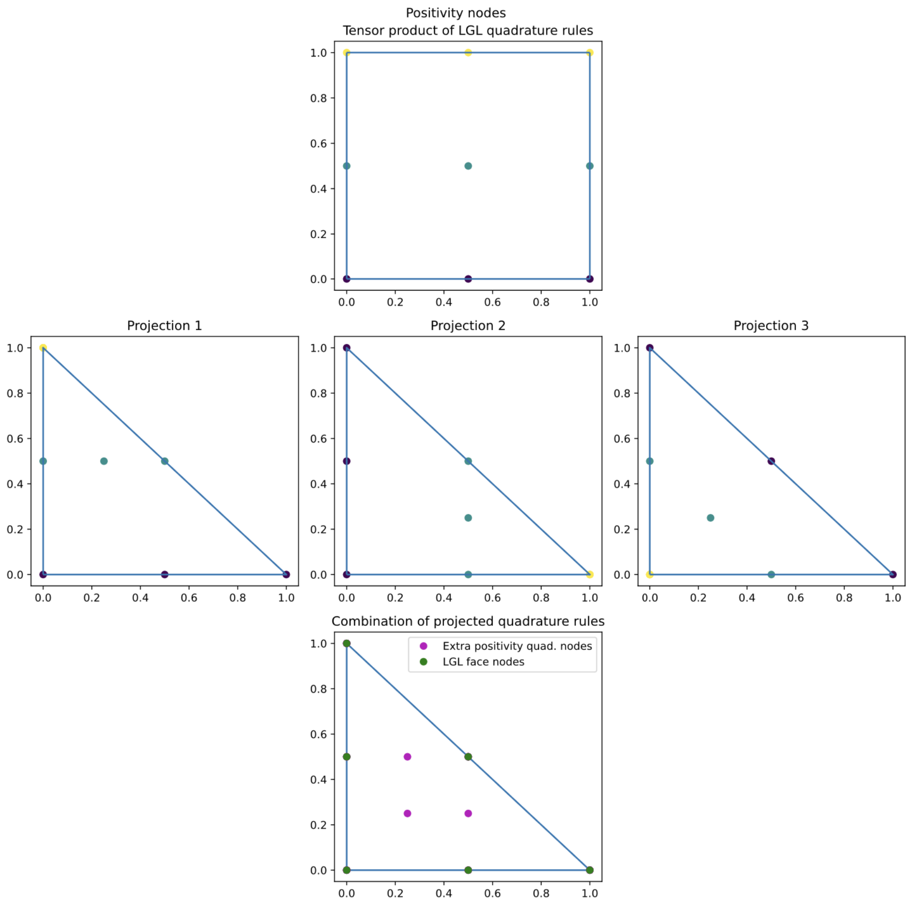

Figure 1: Construction of the extra positivity-preserving quadrature nodes for the third-order triangular basis.