

Efficient basis derivatives evaluations for high-order discontinuous Galerkin schemes

Micaiah Smith-Pierce ^{*1} Aliraza Mazaheri ^{†2}

¹Daniel Guggenheim School of Aerospace Engineering,
Georgia Institute of Technology, Atlanta, GA 30332

²Aerothermodynamics Branch, Research Directorate,
NASA Langley Research Center, Hampton, VA 23681

AIAA AVIATION Forum, June 2020

This material is declared a work of the U.S. Government and is
not subject to copyright protection in the United States.

*Corresponding author, Undergraduate student, micaiah@gatech.edu

†alireza.mazaheri@nasa.gov

Objective

- Compute a basis for the space of K th-degree multivariate polynomials on the following reference elements:

- Line 

- Triangle 

- “Quadrangle” (aka Quadrilateral) 

- Tetrahedral 

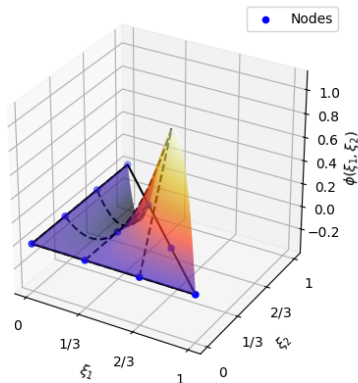
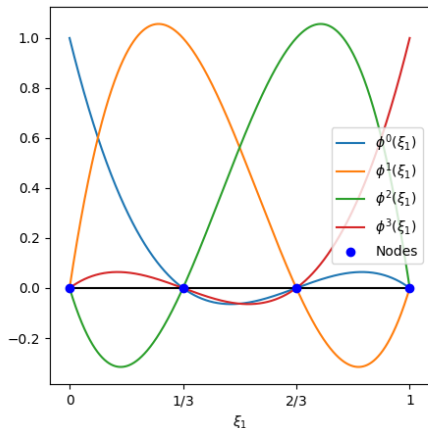
- Hexahedral 

- Prism 

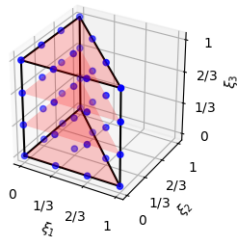
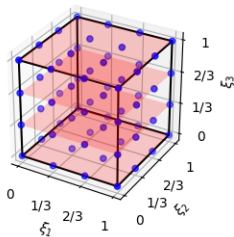
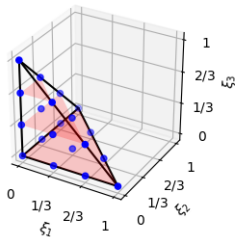
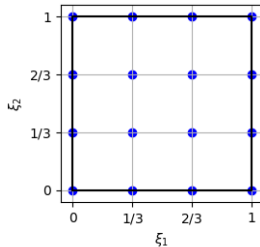
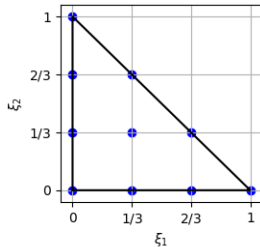
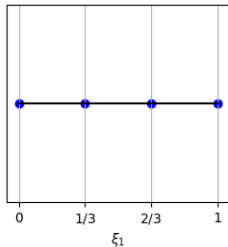
- In this study, the basis consists of Lagrange polynomials.
- We expect some results to generalize to other types of basis.
- Compute ALL nonvanishing derivatives (e.g. $\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial x \partial y}, \dots$)
- Do this in as little time as possible.

Lagrange Basis

- Defined on a set of nodes $\{\xi^i : 1 \leq i \leq N_{dof}\}$
 $N_{dof} :=$ Number of degrees of freedom. $\xi = (\xi_1, \xi_2, \dots, \xi_{N_c}) :=$ local coordinate vector.
- $\phi^i(\xi^j) = \delta_{ij}$



Nodes



Each coordinate of each node is n/K for some $n \in \{0, 1, 2, \dots, K\}$.

Basis definition (1 of 2)

- Line:

$$\phi_{line}^i(\xi_1) = \prod_{\substack{m=0 \\ m \neq p_1(i)}}^K \frac{\xi_1 - m/K}{p_1(i)/K - m/K}$$

- Triangle ($N_{coord} = 3$) and Tetrahedral ($N_{coord} = 4$):

$$\phi_{simplex, N_{coord}}^i(\xi_1, \dots, \xi_{N_{coord}}) = \prod_{d=1}^{N_{coord}} \prod_{m=0}^{p_d(i)-1} \frac{\xi_d - m/K}{p_d(i)/K - m/K}$$

- $p_d : \mathbb{N} \rightarrow \mathbb{N}$ is a “power index” function. See the paper for details.

Basis definition (2 of 2)

- Quadrangle ($N_{coord} = 2$) and Hexahedral ($N_{coord} = 3$):

$$\phi_{brick, N_{coord}}^i(\xi_1, \dots, \xi_{N_{coord}}) = \prod_{d=1}^{N_{coord}} \phi_{line}^{p_d(i)+1}(\xi_d)$$

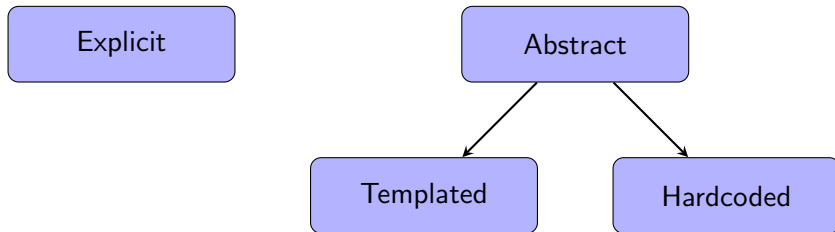
- Prism:

$$\phi_{prism}^i(\xi_1, \xi_2, \xi_3, \xi_4) = \phi_{triangle}^j(\xi_1, \xi_2, \xi_3) \phi_{line}^{p_d(i)+1}(\xi_4)$$

where j is defined implicitly by

$$(p_d)_{triangle}(j) = (p_d)_{prism}(i) \quad : \quad 1 \leq d \leq 3 = N_{coord, triangle}.$$

Solution Approaches



Explicit Approach

- Python preprocessor generates C++ header files.
- Example usage: “python GenerateLagrange.py Tetrahedral 3”
Generates code for 3rd-order tetrahedral. Excerpt from the output:

```
double * getPhi( Coord3D x )
{
    _phi[ 0 ] = 4.5 * (1. - (x[ X1 ] + x[ X2 ] + x[ X3 ]))
                * (x[ X1 ] + x[ X2 ] + x[ X3 ] - 2./3.)
                * (x[ X1 ] + x[ X2 ] + x[ X3 ] - 1./3.);

    :
}
```

- Uses Sympy to expand products.
- Sympy takes symbolic derivatives and performs algebraic simplification.
- Pro: Simplest possible C++ code.
- Con: Expressions get very long very quickly. E.g. 3rd-order Hex involves over 14000 lines.
This can result in unreasonably long compile time.

Abstract Approach (1 of 2)

- Use algorithms to minimize FLOPs.
- Store intermediate results that can be used again.
- Express basis in terms of monomials (e.g. $x, y, x^2, xy, \dots$) so derivatives become trivial.

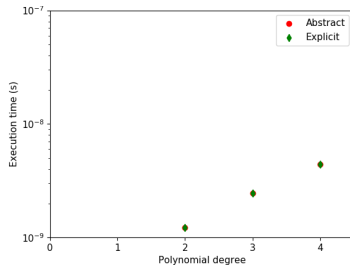
Abstract Approach (2 of 2)

- **Templated approach:** Express algorithms concisely in class templates with order as parameter.
 - No preprocessing required.
 - Fewer lines $\implies$ faster to compile.
 - Harder for compiler to optimize $\implies$ slower to execute.
- **Hardcoded approach:** Use Python preprocessor to hardcode the algorithms for each order.

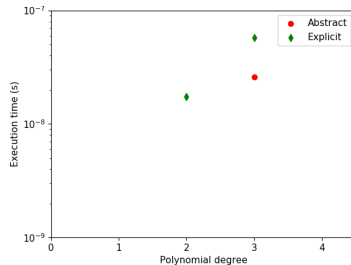
Timing Example: Explicit vs. Abstract Hardcoded

Basis

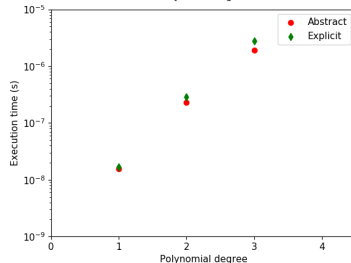
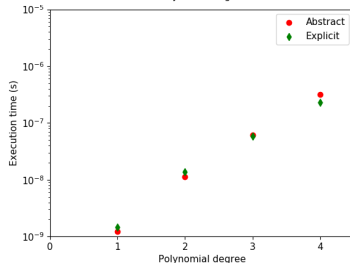
Tetrahedral



Hexahedral



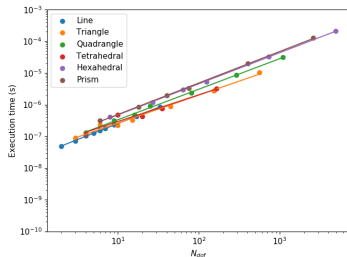
Derivatives



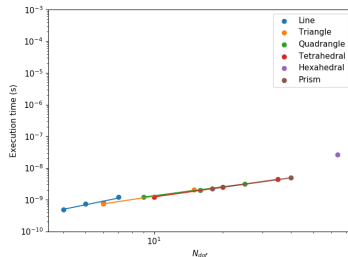
Timing: Asymptotic Behavior

Basis

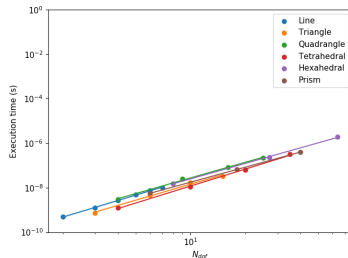
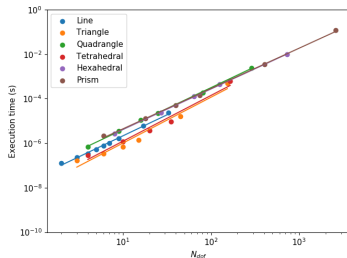
Abstract Templated



Abstract Hardcoded



Derivatives



- Explicit and Abstract techniques were investigated to develop an efficient method for basis and derivative evaluations.
- The naive abstract approach was shown to take orders of magnitude longer execution time compared to the Explicit approach.
- The hardcoded abstract approach was developed and shown to provide a more manageable compilation time and yet explicit-like execution time.
- Future work:
 - Orthogonal basis.
 - Pyramid elements.