Recent results on Bernstein polynomials

Robert Kirby[1]

[1]Baylor University

17 June 2014

BAYLOR UNIVERSITY

Motivation

Bernstein polynomials

Re-envisioning FIAT

Current status

Where we are

- FIAT: All[1] the elements
- FFC: All the forms?
- *But*: not all the algorithms:
  - Lack of "spectral elements" limits high-order efficiency
  - Still hard to do "hard" elements (arbitrary pullbacks)

[1] Affine simplices, that is

Problems for high order

Very large element matrices

$$A_{ij} = \int_K w \nabla \phi_i \cdot \nabla \phi_j \, dx$$

	Standard	Tensor product
Basis size:	$\mathcal{O}(n^d)$	
Element matrix size:	$\mathcal{O}(n^{2d})$	
Cost of local matvec:	$\mathcal{O}(n^{2d})$	$\mathcal{O}(n^{d+1})$

BAYLOR UNIVERSITY

But how do we go fast?

Tensor Products

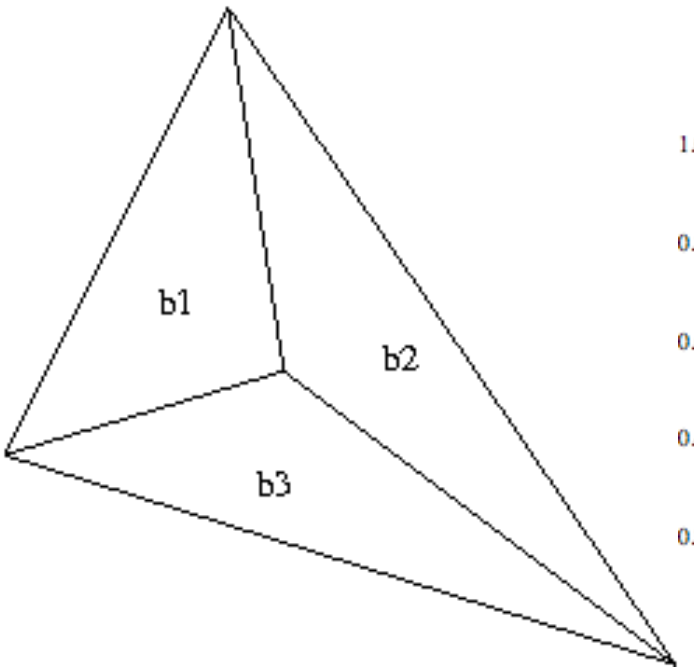
- Work direction-by-direction
- Sum factorization – fast matvecs
- Operation count: $\mathcal{O}(n)$ per entry
- Memory usage: $\mathcal{O}(n^d)$

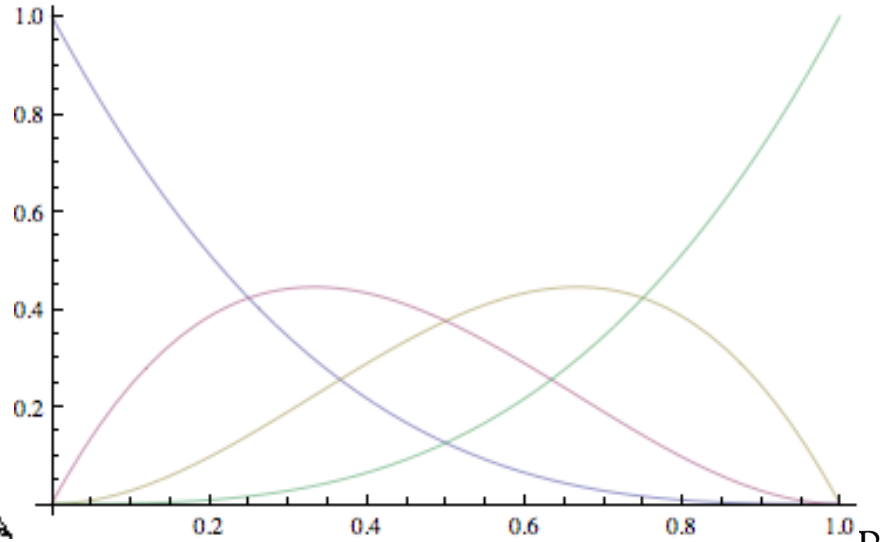
Simplex?

- Collapsed-coordinates: Karniadakis & Sherwin for H^1
- General elements: FIAT (RCK), FEMSTER (White, Castillo)

Bernstein polynomials

$$\left\{ \binom{n}{\alpha} \prod_i b_i^{\alpha_i} \right\}_{|\alpha|=n}$$





BAYLOR UNIVERSITY

Differentiation

It's *sparse* in B-form

$$\frac{\partial}{\partial x} = \sum_{i=1}^{d+1} \frac{\partial b_i}{\partial x} \frac{\partial}{\partial b_i}.$$

Differentiation

It's *sparse* in B-form

$$\frac{\partial}{\partial x} = \sum_{i=1}^{d+1} \frac{\partial b_i}{\partial x} \frac{\partial}{\partial b_i}.$$

$$\frac{\partial}{\partial b_i} B_\alpha^n = \begin{cases} 0, & \alpha_i = 0 \\ \alpha_i B^{n-1}_{\alpha - e_i^d}, & \alpha_i \neq 0 \end{cases}$$

Differentiation

It's *sparse* in B-form

$$\frac{\partial}{\partial x} = \sum_{i=1}^{d+1} \frac{\partial b_i}{\partial x} \frac{\partial}{\partial b_i}.$$

$$\frac{\partial}{\partial b_i} B_\alpha^n = \begin{cases} 0, & \alpha_i = 0 \\ \alpha_i B_{\alpha - e_i^d}^{n-1}, & \alpha_i \neq 0 \end{cases}$$

$D \leftrightarrow$ sparse matrix with at most $d+1$ nonzeros per row

Bernstein polynomials

Some history

- Approximation theory: Bernstein, quasi-interpolants, simplicial splines
- CAGD: stable and fast algorithms for curves/surfaces
- Finite element analysis?
  - Peterson *et. al.*
  - Schumaker (splines)
  - NURBS - Hughes *et. al.*
  - FEEC (Arnold, Falk, Winther)
  - RCK & Ainsworth

Duffy transforms and tensor products

Map d-cube $[0,1]^d$ into d-simplex

Define inductively:

$$\lambda_0 = t_1$$

$$\lambda_i = t_{i+1}\left(1 - \sum_{j=0}^{i-1} \lambda_j\right)$$

$$\lambda_n = 1 - \sum_{j=0}^{n-1} \lambda_j$$

Tensorialize Bernstein

With

$$\mathbf{x}(\mathbf{t}) = \sum_{i=0}^{n} \mathbf{x}_i \lambda_i(\mathbf{t}),$$

we have

$$B_\alpha^r(\mathbf{x}(\mathbf{t})) = \prod_{i=0}^{n} B_{\alpha_i}^{r-\sum_{j=0}^{i}\alpha_j}(t_i)$$

BAYLOR UNIVERSITY

What operations are fast?

Evaluation

Given $u = \sum_{|\alpha|=n} u_\alpha B^n_\alpha$,

$$\{u_\alpha\}_\alpha \mapsto \{u(\xi_q)\}_q ,$$

when $\{\xi_q\}_q$ are Stroud points. Requires $\mathcal{O}(n^{d+1})$ and *no* pre-tabulated data.

Moment computation

Given $\{f_q = f(\xi_q)\}_q$

$$\{f_q\} \mapsto \left\{ \int_T f B^n_\alpha dx \right\}_\alpha$$

requires $\mathcal{O}(n^{d+1})$ and *no* pre-tabulated data.

Derivatives?

Evaluate/integrate followed by *short* linear combinations!

BAYLOR UNIVERSITY

Optimal-complexity assembly

Constant-order work per entry

Since $B^r_\alpha B^s_\beta = \frac{\binom{\alpha+\beta}{\alpha}}{\binom{r+s}{r}} B^{r+s}_{\alpha+\beta}$, so matrix formation

$$M_{\alpha\beta} = \int_T f B^r_\alpha B^s_\beta$$

just requires (plus bookkeeping) all moments

$$\left\{ \int_T f B^{r+s}_\gamma dx \right\}_\gamma$$

The de Rham complex

FEEC (Arnold, Falk, Winther)

Basis functions for $P_n^- \Lambda^1$: $B_\alpha^{n-1} \phi_{ij}$
Basis functions for $P_n^- \Lambda_2$: $B_\alpha^{n-1} \phi_{ijk}$, where

$$\phi_{ij} = b_i db_j - b_j db_i$$
$$\phi_{ijk} = b_i db_j \wedge db_k - b_j db_i \wedge db_k + b_k d\lambda_i \wedge db_j$$

BAYLOR UNIVERSITY

Convert to Bernstein form

Short linear combination

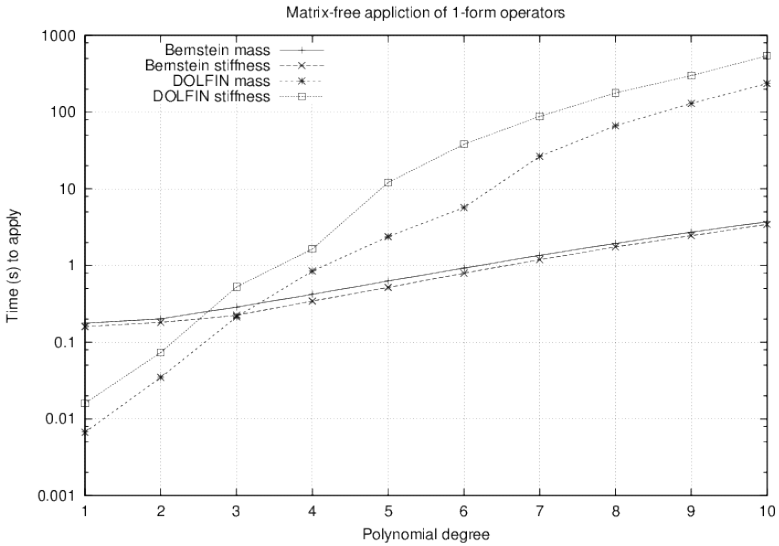
$$
\begin{aligned}
B_\alpha^{n-1}\phi_{ij} &= b_i B_\alpha^{n-1} db_j - b_j B_\alpha^{n-1} db_i \\
&= b_i \frac{(n-1)!}{\alpha!}\mathbf{b}_d^\alpha db_j - b_j \frac{(n-1)!}{\alpha!}\mathbf{b}_d^\alpha db_i \\
&= \frac{(n-1)!}{\alpha!}\mathbf{b}_d^{\alpha+e_i} db_j - \frac{(n-1)!}{\alpha!}\mathbf{b}_d^{\alpha+e_j} db_i \\
&= \frac{\alpha_i+1}{n} B_{\alpha+e_i}^n db_j - \frac{\alpha_j+1}{n} B_{\alpha+e_j}^n db_i
\end{aligned}
$$

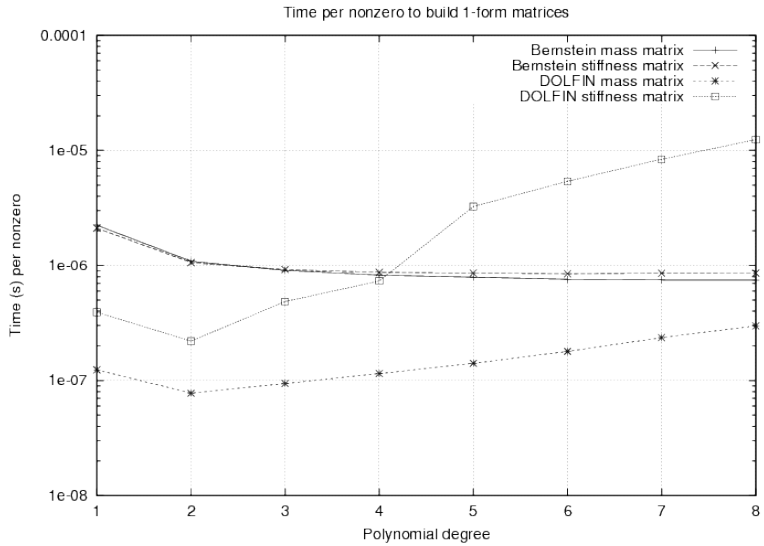
Algorithms

Conversion

- Each k-form basis function requires $k+1$ Bernstein polynomials
- Operator formation/application requires fast evaluation/integration kernels for Bernstein, plus some simple linear algebra.
- Optimal complexity for $H(\mathrm{div})$ and $H(\mathrm{curl})$.

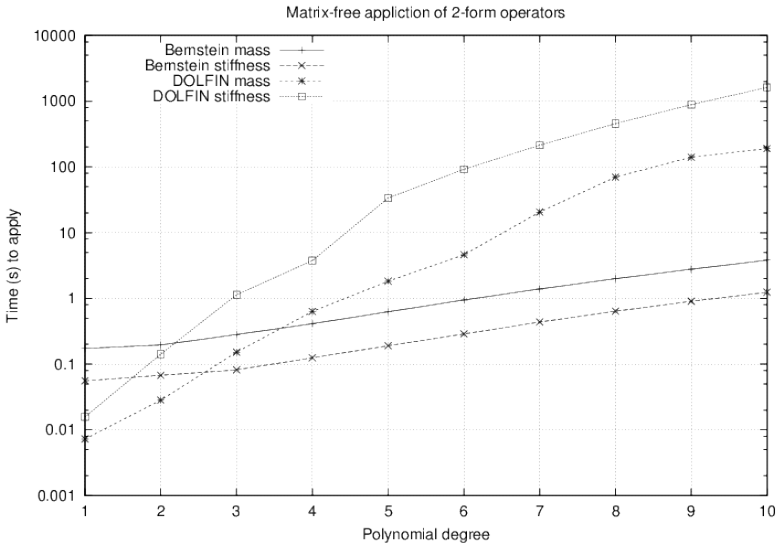
1-form action and build time, per nonzero

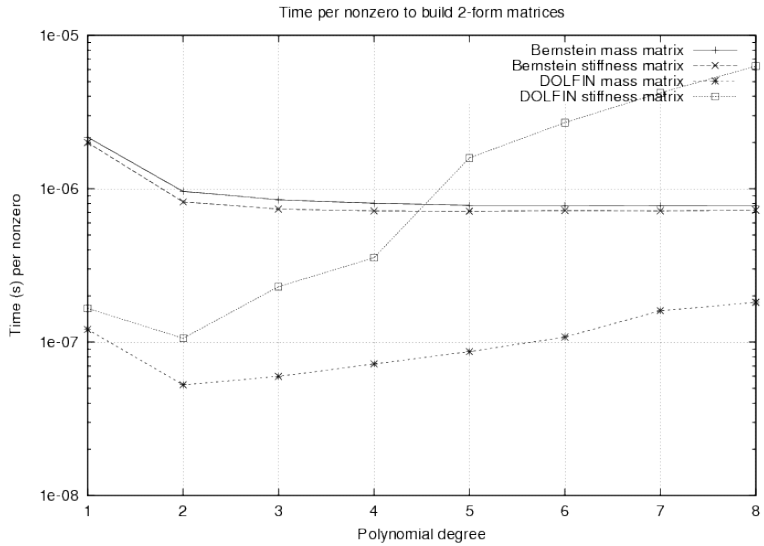




BAYLOR UNIVERSITY

2-form action and build time, per nonzero





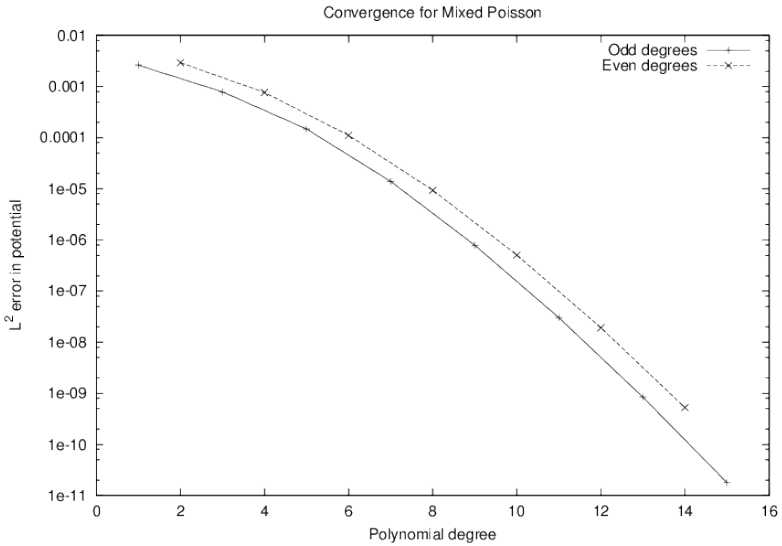
BAYLOR UNIVERSITY

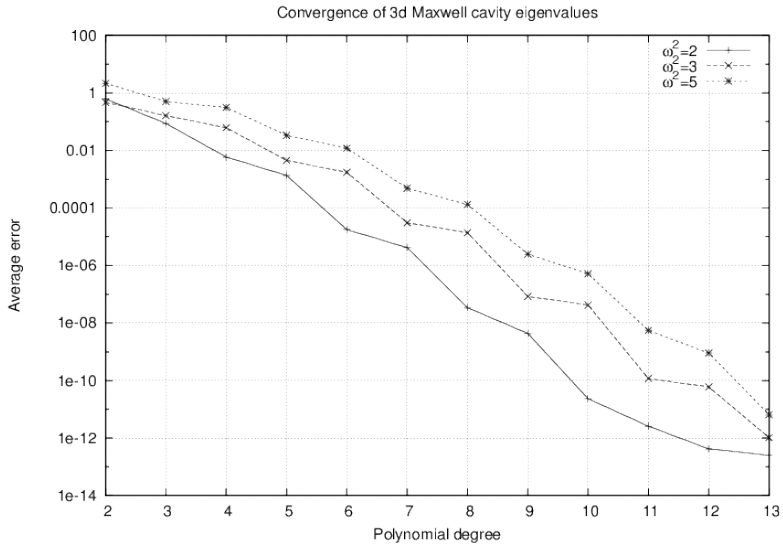
FEniCS Code generation/compilation (tensor mode)

Times are in *hours*

k	1-form mass	1-form stiffness	2-form mass	2-form stiffness
1	4.8e-4	5.3e-4	5.2e-4	5.4e-4
2	8.4e-4	7.7e-4	7.5e-4	7.1e-4
3	7.6e-3	1.7e-3	3.6e-3	1.5e-3
4	7.2e-2	3.0e-3	3.5e-2	2.5e-3
5	2.7e-1	8.8e-3	2.9e-1	7.0e-3
6	2.0e0	2.9e-2	9.4e-1	2.1e-2
7	8.5e0	6.9e-2	5.8e0	5.4e-2
8	3.7e1	1.9e-1	2.7e1	1.5e-1
9	1.2e2	5.8e-1	8.7e1	4.0e-1

Accuracy?





BAYLOR UNIVERSITY

How this breaks FIAT/FFC

Let me count the ways

- Algorithmic granularity
- No `dmats`
- No orthogonal basis
- No pullback[2]
- Need to generate completely different code!

[2] Or at least, it's not necessary

It's been ten years...

What I got right

- We *can* do lots of elements.
- We *can* mathematize FEM code.
- `eids` (plus some helpers)

And what's missing

- Rigid compile/run-time break:
  - Run-time evaluation?
  - General pullbacks?
  - `(quote elem)`?
- General pullbacks
- Tensor-product elements
- `dmats` and `V` suboptimal.

BAYLOR UNIVERSITY

Going forward

What "elements" know

- How to tabulate values and basis functions
- Association of dof to facets
- *Structure*:
  - Tensor products
  - Fast algorithms
  - "Preferred" quadrature schemes
  - Transformation information

Question: How does the element tell this to the form compiler?

What the form compiler knows

- Understand (UFL) AST
- Query elements for:
  - Pullback schemes (default de Rham *or* "weird" elements)
  - Evaluation/integration recipes
- Schedule, create temporaries, batching, vectorization, etc...
- Build the (code) AST
- No form compiler edits for new elements/transforms!