FIAT: from basis functions to efficient finite element solvers

Pablo D. Brubeck Patrick E. Farrell Robert C. Kirby (Baylor)

University of Oxford

November 12, 2024

Part 1

The FIAT paradigm

FIAT (the FInite element Automatic Tabulator) is the Firedrake component that tabulates finite element bases on quadrature points.

FIAT can be used by general clients, and is used by legacy FEniCS.

Aim for today: to show you how FIAT implements complicated finite elements with highly desirable properties.

FIAT uses Ciarlet's definition of a finite element

A finite element is a triple $(K, P, \mathcal{L})$ where

- K is a cell (simplex, tensor product, or simplicial complex),
- P is the local function space,
- $\mathcal{L} = \{\ell_i\}$ is a basis of functionals spanning the dual space P^*.

FIAT uses Ciarlet's definition of a finite element

A finite element is a triple $(K, P, \mathcal{L})$ where

- K is a cell (simplex, tensor product, or simplicial complex),
- P is the local function space,
- $\mathcal{L} = \{\ell_i\}$ is a basis of functionals spanning the dual space P^*.

We denote $\{\ell_i\}$ as the degrees of freedom, and they glue together across elements to give the right continuity (H^1, $H(\mathrm{div})$, etc.)

- $\ell_i(v) = v(x)$: evaluation at a point,
- $\ell_i(v) = \mathbf{s} \cdot \nabla v(x)$: directional derivative at a point,
- $\ell_i(v) = \int_f vq \, \mathrm{d}s$: integral moment on a facet (quadrature),
- etc.

FIAT constructs the **primal basis** $\{\phi_j\}$ from a set of degrees of freedom $\{\ell_i\}$, satisfying $\ell_i(\phi_j) = \delta_{ij}$.

FIAT constructs the **primal basis** $\{\phi_j\}$ from a set of degrees of freedom $\{\ell_i\}$, satisfying $\ell_i(\phi_j) = \delta_{ij}$.

We construct $\{\phi_j\}$ from a set $\{q_k\}$ spanning P (e.g. orthonormal polynomials),

$$\phi_j = \sum_k A_{jk} q_k.$$

FIAT constructs the **primal basis** $\{\phi_j\}$ from a set of degrees of freedom $\{\ell_i\}$, satisfying $\ell_i(\phi_j) = \delta_{ij}$.

We construct $\{\phi_j\}$ from a set $\{q_k\}$ spanning P (e.g. orthonormal polynomials),

$$\phi_j = \sum_k A_{jk} q_k.$$

The Kronecker property leads to the **generalized Vandermonde system**,

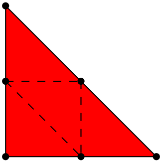
$$\ell_i(\phi_j) = \delta_{ij} = \sum_k A_{jk} \ell_i(q_k).$$

The matrix of expansion coefficients is then $A = V^{-\top}$, where $V_{ij} = \ell_i(q_j)$.

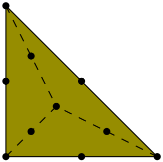
Part 2

Macroelements

Here are some macroelements.

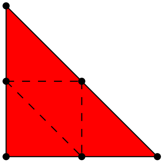


$\mathbb{P}_1^{\text{iso}} - \mathbb{P}_2$

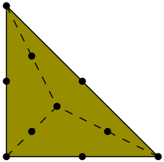


$\mathbb{P}_2$-Alfeld

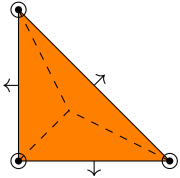
Here are some macroelements.



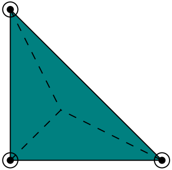
$\mathbb{P}_1^{\text{iso}} - \mathbb{P}_2$



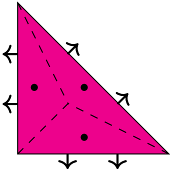
$\mathbb{P}_2$-Alfeld



HCT



Reduced HCT



Johnson–Mercier

FIAT macroelements are implemented as Ciarlet triples.

1. Generalize cell types to **simplicial complexes**, and implement different splittings (Iso, Alfeld, Powell-Sabin).

FIAT macroelements are implemented as Ciarlet triples.

1. Generalize cell types to **simplicial complexes**, and implement different splittings (Iso, Alfeld, Powell-Sabin).

2. Define **piecewise polynomial spaces** on simplicial complexes, with constraints that impose the relevant continuity (C^k, or normal continuity).

FIAT macroelements are implemented as Ciarlet triples.

1. Generalize cell types to **simplicial complexes**, and implement different splittings (Iso, Alfeld, Powell-Sabin).
2. Define **piecewise polynomial spaces** on simplicial complexes, with constraints that impose the relevant continuity (C^k, or normal continuity).
3. Specify the **degrees of freedom** on either the entities of split cell or the unsplit parent cell entities.

FIAT macroelements are implemented as Ciarlet triples.

1. Generalize cell types to **simplicial complexes**, and implement different splittings (Iso, Alfeld, Powell-Sabin).

2. Define **piecewise polynomial spaces** on simplicial complexes, with constraints that impose the relevant continuity (C^k, or normal continuity).

3. Specify the **degrees of freedom** on either the entities of split cell or the unsplit parent cell entities.

Construct composite quadrature rules on the split cell.

Why Macroelements?

- To achieve the lowest order possible.

Why Macroelements?

- To achieve the lowest order possible.

Space	Lowest p macro	Lowest p non-macro
$H(\mathrm{div}, \mathrm{sym})$ (2D)	1 (JM)	3 (AW)
H^1 div-free (2D)	2 (Alfeld)	4 (SV)
H^2 (2D)	3 (HCT)	5 (Argyris)
H^1 div-free (3D)	3 (Alfeld)	6 (SV)

Why Macroelements?

- To achieve the lowest order possible.

Space	Lowest p macro	Lowest p non-macro
$H(\mathrm{div}, \mathrm{sym})$ (2D)	1 (JM)	3 (AW)
H^1 div-free (2D)	2 (Alfeld)	4 (SV)
H^2 (2D)	3 (HCT)	5 (Argyris)
H^1 div-free (3D)	3 (Alfeld)	6 (SV)

- But I like high-order.

Why Macroelements?

- To achieve the lowest order possible.

Space	Lowest p macro	Lowest p non-macro
$H(\mathrm{div}, \mathrm{sym})$ (2D)	1 (JM)	3 (AW)
H^1 div-free (2D)	2 (Alfeld)	4 (SV)
H^2 (2D)	3 (HCT)	5 (Argyris)
H^1 div-free (3D)	3 (Alfeld)	6 (SV)

- But I like high-order.
  - Cheaper low-order coarse spaces.
  - Equivalent low-order-refined preconditioners for high-order.

Why Macroelements?

- To achieve the lowest order possible.

Space	Lowest p macro	Lowest p non-macro
$H(\mathrm{div}, \mathrm{sym})$ (2D)	1 (JM)	3 (AW)
H^1 div-free (2D)	2 (Alfeld)	4 (SV)
H^2 (2D)	3 (HCT)	5 (Argyris)
H^1 div-free (3D)	3 (Alfeld)	6 (SV)

- But I like high-order.
  - Cheaper low-order coarse spaces.
  - Equivalent low-order-refined preconditioners for high-order.

- To avoid extra inter-element continuity (super-smoothness).

Why Macroelements?

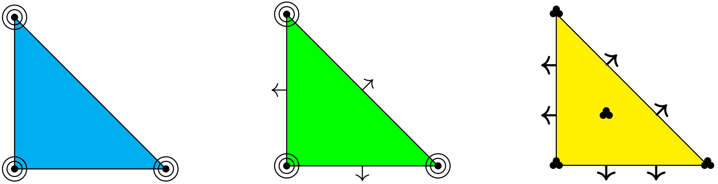
- To achieve the lowest order possible.

Space	Lowest p macro	Lowest p non-macro
$H(\mathrm{div}, \mathrm{sym})$ (2D)	1 (JM)	3 (AW)
H^1 div-free (2D)	2 (Alfeld)	4 (SV)
H^2 (2D)	3 (HCT)	5 (Argyris)
H^1 div-free (3D)	3 (Alfeld)	6 (SV)

- But I like high-order.
  - Cheaper low-order coarse spaces.
  - Equivalent low-order-refined preconditioners for high-order.

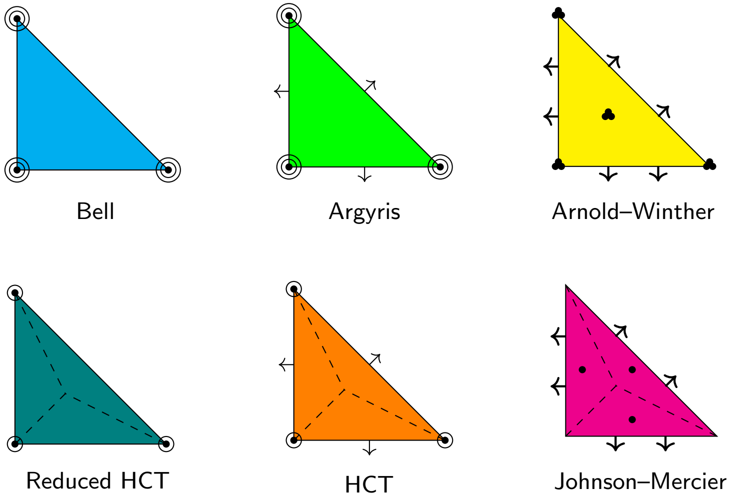
- To avoid extra inter-element continuity (super-smoothness).
  - Discretizations of H^2 and $H(\mathrm{div}, \mathrm{sym})$ are significantly easier to implement in 3D with macroelements.

Avoiding super-smoothness: save 3 DOFs per vertex

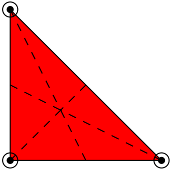


Bell   Argyris   Arnold–Winther

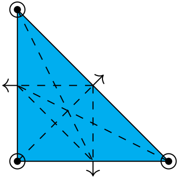
Avoiding super-smoothness: save 3 DOFs per vertex



C^1 triangles (C^1 tetrahedron is WIP)

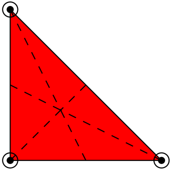


Quadratic Powell-Sabin 6

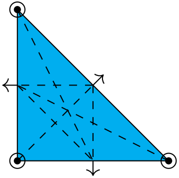


Quadratic Powell-Sabin 12

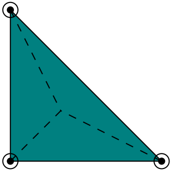
C^1 triangles (C^1 tetrahedron is WIP)



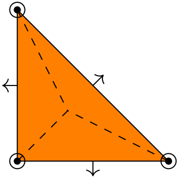
Quadratic Powell-Sabin 6



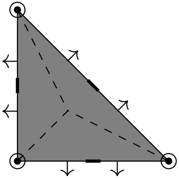
Quadratic Powell-Sabin 12



Reduced HCT



HCT_3



HCT_4

The biharmonic PDE solved with (non-nested) multigrid.

```
from firedrake import *
mh = MeshHierarchy(UnitSquareMesh(4, 4), 3)
mesh = mh[-1]

V = FunctionSpace(mesh, "HCT-red", 3)
u = Function(V)
v = TestFunction(V)
F = (inner(grad(grad(u)), grad(grad(v)))*dx
     - inner(1, v)*dx)

# Clamped bcs: u = du/dn = 0
bcs = [DirichletBC(V, 0, "on_boundary")]
solve(F == 0, u, bcs=bcs, solver_parameters={
   "snes_type": "ksponly",
   "ksp_monitor": None,
   "ksp_type": "cg",
   "pc_type": "mg",
   "mg_levels_pc_type": "jacobi",
})  # 6-digit residual reduction in 10 V-cycles
```

Stokes flow with lowest order (try this at home!)

```
from firedrake import *
mesh = UnitSquareMesh(4, 4)

# CG/DG Macroelement variants (works for any degree/cell type!)
V = VectorFunctionSpace(mesh, "CG", 2, variant="alfeld")
Q = FunctionSpace(mesh, "DG", 1, variant="alfeld")
Z = V * Q

# Incompressible Stokes flow
z = Function(Z)
u, p = split(z)
v, q = TestFunctions(Z)
F = (inner(grad(u), grad(v))*dx - inner(p, div(v))*dx
     - inner(div(u), q)*dx)
bcs = [DirichletBC(Z.sub(0), 0, (1, 2, 3)),
       DirichletBC(Z.sub(0), Constant([1, 0]), (4,))]
solve(F == 0, z, bcs=bcs)
print("Divergence error", norm(div(u)))  # 3.8E-15
```

Stokes elements

One of the biggest challenges in finite elements was the construction of low-order, H^1-conforming, inf-sup stable, and divergence-free elements for Stokes flow.

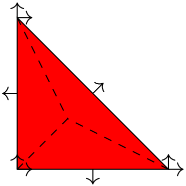
Element	Lowest p	conforming	inf-sup	div-free
Raviart-Thomas	$[0, 1)$	✗	✓	✓
Bernardi-Raugel	$[1, d)$	✓	✓	✗
Taylor-Hood	2	✓	✓	✗
SV non-macro	$2d$	✓	✓	✓
SV macro	d	✓	✓	✓

Stokes macroelements in 2D

The *Stokes complex* gives velocity elements from stream function elements

$$\mathrm{HCT}^{\mathrm{red}} \xrightarrow{\mathrm{curl}} P \xrightarrow{\mathrm{div}} \mathbb{P}_0$$

$$P = \mathrm{curl}(\mathrm{HCT}^{\mathrm{red}}) + \mathbf{x}\,\mathbb{P}_0$$



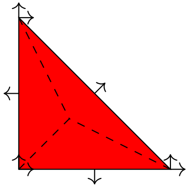
Arnold–Qin

Stokes macroelements in 2D

The *Stokes complex* gives velocity elements from stream function elements

$$\mathrm{HCT}^{\mathrm{red}} \xrightarrow{\mathrm{curl}} P \xrightarrow{\mathrm{div}} \mathbb{P}_0$$

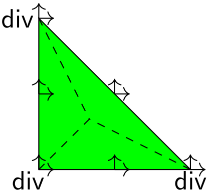
$$P = \mathrm{curl}(\mathrm{HCT}^{\mathrm{red}}) + \mathbf{x}\,\mathbb{P}_0$$



Arnold–Qin

$$\mathrm{HCT} \xrightarrow{\mathrm{curl}} P \xrightarrow{\mathrm{div}} C^0\mathbb{P}_1(K_r)$$

$$P = \mathrm{curl}(\mathrm{HCT}) + \mathbf{x}\,C^0\mathbb{P}_1(K_r)$$



Alfeld–Sorokina

Stokes macroelements in any dimension

The stream function element is not always available.

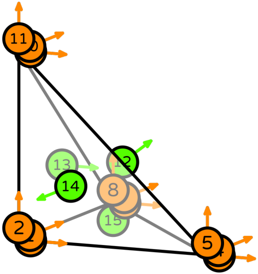
$$C^1 \xrightarrow{\text{grad}} ?? \xrightarrow{\text{curl}} P \xrightarrow{\text{div}} \mathbb{P}_0$$

Stokes macroelements in any dimension

The stream function element is not always available.

$$C^1 \xrightarrow{\text{grad}} ?? \xrightarrow{\text{curl}} P \xrightarrow{\text{div}} \mathbb{P}_0$$

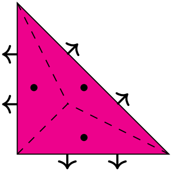
Guzman & Neilan (2018) give Stokes macroelements in any dimension by enriching a simpler space with div-free face bubbles (FB) on an Alfeld split.



- $P = [\mathbb{P}_1]^d +$ FB.
- $P =$ Alfeld–Sorokina $+$ FB.

Constructing symmetric-tensor valued, $H(\mathrm{div})$-conforming discretizations for stress-displacement formulations of elasticity has been another long-stading challenge in finite elements.

The Johnson–Mercier macroelement offers a simpler alternative to non-macroelements, and it is much easier to implement.



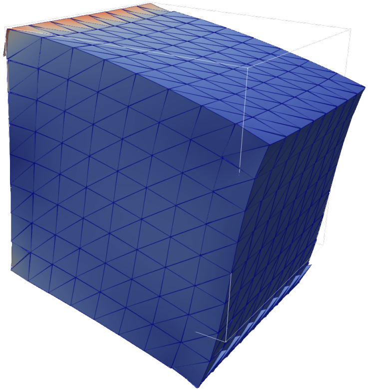
Johnson–Mercier

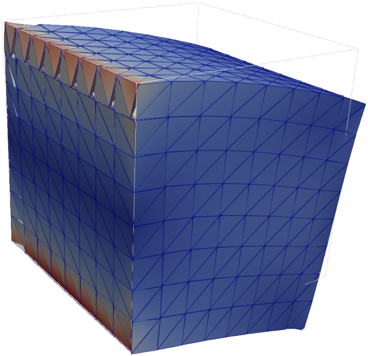
Mixed problems with split and unsplit elements.

```python
from firedrake import *
mesh = UnitCubeMesh(8, 8, 8)

# Stress-displacement formulation of linear elasticity
S = FunctionSpace(mesh, "JM", 1)
V = VectorFunctionSpace(mesh, "DG", 1)
Z = S * V

z = Function(Z)
sig, u = split(z)
tau, v = TestFunctions(Z)
F = ((inner(sig, tau) - (1/3)*tr(sig)*tr(tau))*dx
     + inner(u, div(tau))*dx
     + inner(div(sig), v)*dx)

# Traction boundary conditions
sigbc = Constant([[0, 0, 0], [0, 0, 0], [-0.1, 0, 0]])
bcs = [DirichletBC(Z.sub(0), 0, (3, 4, 5, 6)),
       DirichletBC(Z.sub(0), sigbc, (2,))]
solve(F == 0, z, bcs=bcs)
File("stress.pvd").write(*z.subfunctions)
```

Mixed problems with split and unsplit elements.

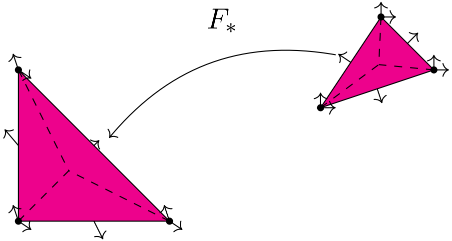
Stress-displacement using $\mathrm{JM}_1 \subset H(\mathrm{div}, \mathrm{sym}) \times \mathrm{DG}_1 \subset L^2$





Transformation of finite elements

Elements that have some normal/tangential or derivative degrees of freedom are not affine-equivalent, and physical basis functions are obtained by carefully recombining reference basis functions.



Pushing forward the HCT derivative nodes in physical space does *not* produce the reference derivative nodes.

Macroelements are great

- FIAT natively supports macroelements.
- Cheaper discretizations for the biharmonic, Stokes, and mixed elasticity.
- If you love low-order, you will love macroelements.

Macroelements are great

- FIAT natively supports macroelements.
- Cheaper discretizations for the biharmonic, Stokes, and mixed elasticity.
- If you love low-order, you will love macroelements.
- If you love high-order, you will love macroelements.

Part 3

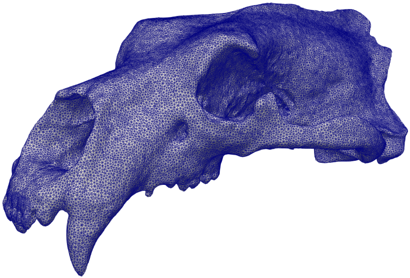
High-order elements on simplices

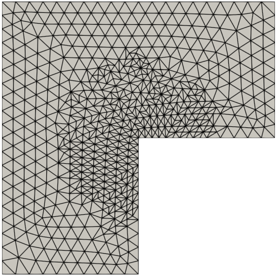
Why high-order FEM on simplices?

High-order finite element discretizations **converge rapidly** and expose **high-arithmetic intensity**.

They expose structure enabling fast operator application via sum-factorization and fast solvers via the **fast diagonalization method** (FDM). This is not so obvious for simplices.

Simplicial meshes offer great **geometric flexibility** and **adaptivity**.





FIAT release paper (on arXiv)

FIAT: improved performance and accuracy for high-order finite elements (B., Kirby, Laakmann & Mitchell, 2024).

FIAT release paper (on arXiv)

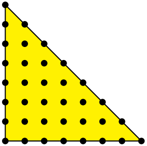
FIAT: improved performance and accuracy for high-order finite elements (B., Kirby, Laakmann & Mitchell, 2024).

- Faster element instantiation and tabulation.
- Cheaper quadrature schemes (Xiao & Gimbutas, 2010).
- Better Lagrange-type degrees of freedom (Isaac, 2020).
- Textbook integral-type degrees of freedom for $H(\mathrm{div})/H(\mathrm{curl})$.

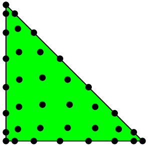
Better conditioned degrees of freedom for Lagrange interpolation

Originally, FIAT only supported non-equispaced 1D GL/GLL elements. Good interpolation points are typically expensive to compute on simplices.

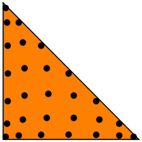
Firedrake now uses the points from (Isaac, 2020) **by default**. These are recursively defined from the 1D GL/GLL families.



Equispaced



Recursive GLL



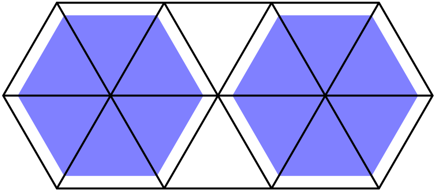
Recursive GL

Fast solvers for high-order discretizations

Consider the discrete Poisson problem $Ax = b$.

Fast solver: Conjugate gradients + p-robust domain decomposition

Construct subdomains around mesh vertices and use the lowest-order element as the coarse space. Only requires $\mathcal{O}(1)$ CG iterations (Schöberl, Melenk, Pechstein & Zaglmayr, 2008).



Two *vertex-star* subdomains, $\star v$

$$V_{h,p} = V_{h,1} + \sum_{v \in \text{vertices}} V_{h,p}\Big|_{\star v}$$

Fast solvers for high-order discretizations

Consider the discrete Poisson problem $Ax = b$.

Fast solver: Conjugate gradients + p-robust domain decomposition

Construct subdomains around mesh vertices and use the lowest-order element as the coarse space. Only requires $\mathcal{O}(1)$ CG iterations (Schöberl, Melenk, Pechstein & Zaglmayr, 2008).

Bottleneck: matrix-based subdomain solvers

A single vertex-star can typically have around 24 tetrahedra. Memory, setup, and solution costs become prohibitively expensive as p increases.

$$V_{h,p} = V_{h,1} + \sum_{v \in \text{vertices}} V_{h,p}\Big|_{\star v}$$

Central challenge

How do you solve the vertex-star problems? They get denser and denser as p increases. ($\mathcal{O}(p^7)$ assembly, $\mathcal{O}(p^9)$ factorization, $\mathcal{O}(p^6)$ application.)

First step

Define new finite elements that give *much sparser* discrete operators, for some problems.

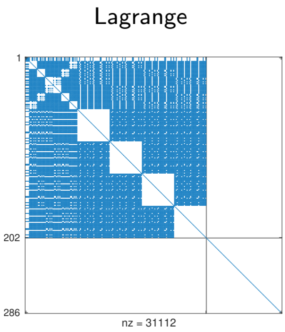
Main contribution

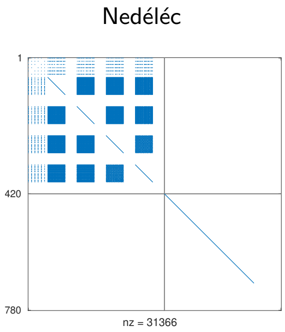
In this talk, we present fast iterative solvers for the *Riesz maps*

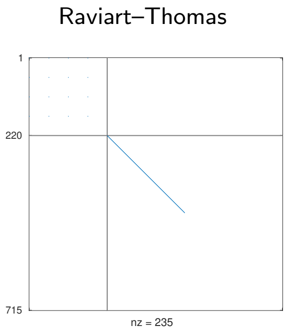
$$
\begin{aligned}
&\text{find } u \in H(\mathrm{grad}) : (v, u) + (\mathrm{grad}\, v, \mathrm{grad}\, u) = (v, f) \quad \forall v \in H(\mathrm{grad})\\
&\text{find } \mathbf{u} \in H(\mathrm{curl}) : (\mathbf{v}, \mathbf{u}) + (\mathrm{curl}\, \mathbf{v}, \mathrm{curl}\, \mathbf{u}) = (\mathbf{v}, \mathbf{f}) \quad \forall \mathbf{v} \in H(\mathrm{curl})\\
&\text{find } \mathbf{u} \in H(\mathrm{div}) : (\mathbf{v}, \mathbf{u}) + (\mathrm{div}\, \mathbf{v}, \mathrm{div}\, \mathbf{u}) = (\mathbf{v}, \mathbf{f}) \quad \forall \mathbf{v} \in H(\mathrm{div})
\end{aligned}
$$

on unstructured triangular/tetrahedral meshes with very high polynomial degree.

New degrees of freedom that decouple the interior and interface.







p-robust solver for the Riesz maps on $\Omega = [0,1]^d$.

- KSP: Conjugate Gradients
  - PC: Two-level vertex-star additive Schwarz
    - p-fine: Interior-facet additive fieldsplit
      - Interior: point-Jacobi
      - Facet: PCASM/PCPATCH
    - p-coarse ($p = 1$): Cholesky

p-robust solver for the Riesz maps on $\Omega = [0,1]^d$.

CG iteration counts (rel. tol. $= 10^{-8}$).

d	p	$H(\mathrm{grad})$	$H(\mathrm{curl})$	$H(\mathrm{div})$
2	4	20	24	24
	7	20	23	23
	10	20	23	23
	14	20	23	23
3	4	23	39	19
	7	23	42	19
	10	23	42	19

Additive space decompositions on the interface Schur complement

- Vertex-star + Lowest-order
- Edge-star + Vertex-star on $\mathrm{grad}\, H(\mathrm{grad})$ + Lowest-order
- Edge-star + Lowest-order

Building a new element for $H(\mathrm{grad}, [-1, 1])$

Standard CG element: degrees of freedom (DOFs) are pointwise evaluations.

Building a new element for $H(\mathrm{grad}, [-1, 1])$

Standard CG element: degrees of freedom (DOFs) are pointwise evaluations.

Key idea for new element

Choose new DOFs to promote orthogonality in $(\cdot,\cdot)_{L^2}$ and $(\cdot,\cdot)_{H(\mathrm{grad})}$.

Building a new element for $H(\mathrm{grad}, [-1, 1])$

Standard CG element: degrees of freedom (DOFs) are pointwise evaluations.

Key idea for new element

Choose new DOFs to promote orthogonality in $(\cdot, \cdot)_{L^2}$ and $(\cdot, \cdot)_{H(\mathrm{grad})}$.

We choose the DOFs as:

- Point evaluation at the vertices ($H(\mathrm{grad})$-conforming)

Building a new element for $H(\mathrm{grad}, [-1, 1])$

Standard CG element: degrees of freedom (DOFs) are pointwise evaluations.

Key idea for new element

Choose new DOFs to promote orthogonality in $(\cdot, \cdot)_{L^2}$ and $(\cdot, \cdot)_{H(\mathrm{grad})}$.

We choose the DOFs as:

- Point evaluation at the vertices ($H(\mathrm{grad})$-conforming)
- Fast diagonalization method (FDM): find $\{\hat{s}_i\}_{i=1:(p-1)} \subset \mathcal{P}_p(\hat{\mathcal{I}})$ s.t.

$$(\hat{s}_i', \hat{s}_j')_{\hat{\mathcal{I}}} = \lambda_i \delta_{ij}, \quad (\hat{s}_i, \hat{s}_j)_{\hat{\mathcal{I}}} = \delta_{ij}, \quad \hat{s}_i(-1) = \hat{s}_i(1) = 0.$$

Define DOFs to be integral moments against these eigenfunctions.

Building a new element for $H(\mathrm{grad}, [-1,1])$

Standard CG element: degrees of freedom (DOFs) are pointwise evaluations.

Key idea for new element

Choose new DOFs to promote orthogonality in $(\cdot,\cdot)_{L^2}$ and $(\cdot,\cdot)_{H(\mathrm{grad})}$.

We choose the DOFs as:

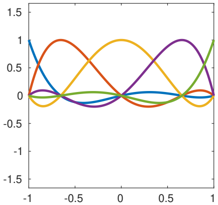
- Point evaluation at the vertices ($H(\mathrm{grad})$-conforming)
- Fast diagonalization method (FDM): find $\{\hat{s}_i\}_{i=1:(p-1)} \subset \mathcal{P}_p(\hat{\mathcal{I}})$ s.t.

$$(\hat{s}_i', \hat{s}_j')_{\hat{\mathcal{I}}} = \lambda_i \delta_{ij}, \quad (\hat{s}_i, \hat{s}_j)_{\hat{\mathcal{I}}} = \delta_{ij}, \quad \hat{s}_i(-1) = \hat{s}_i(1) = 0.$$

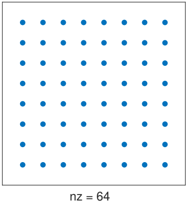
  Define DOFs to be integral moments against these eigenfunctions.

For 1D Poisson and mass matrices, the interior-interior block is diagonal!

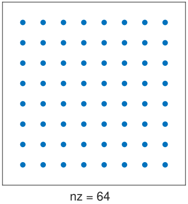
1D stiffness and mass are sparse in the FDM basis.



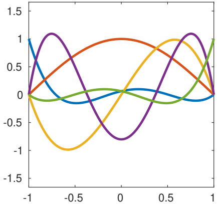
(a) `variant="gll"`



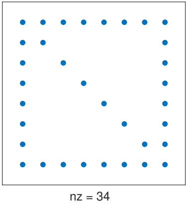
(b) $\hat{A}$ – `"gll"`



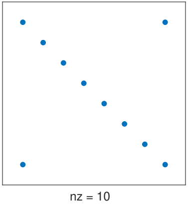
(c) $\hat{B}$ – `"gll"`



(d) `variant="fdm"`

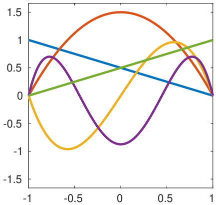


(e) $\hat{A}$ – `"fdm"`

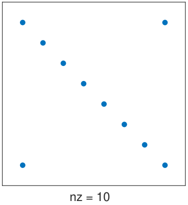


(f) $\hat{B}$ – `"fdm"`

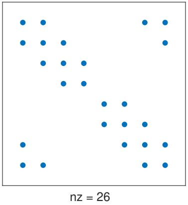
1D stiffness and mass are sparse in the FDM basis.



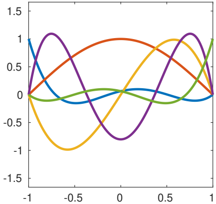
(a) `variant="hier"`



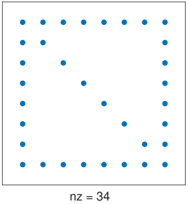
(b) $\hat{A}$ – `"hier"`



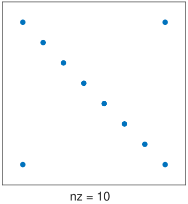
(c) $\hat{B}$ – `"hier"`



(d) `variant="fdm"`



(e) $\hat{A}$ – `"fdm"`

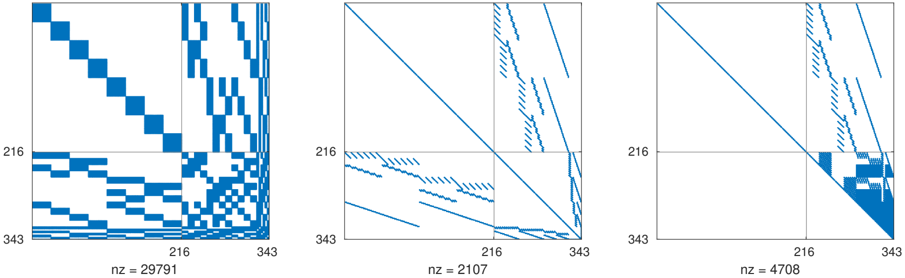


(f) $\hat{B}$ – `"fdm"`

The fast diagonalization method on tensor-product cells

On the interior of a quad/hex, the FDM basis is the discrete analogue of the eigenbasis in the method of separation of variables.

$$A = \begin{cases} B_y \otimes A_x + A_y \otimes B_x & d = 2, \\ B_z \otimes B_y \otimes A_x + B_z \otimes A_y \otimes B_x + A_z \otimes B_y \otimes B_x & d = 3. \end{cases}$$



(a) A_{grad}, standard (b) A_{grad}, FDM (c) $\mathrm{chol}(A_{\mathrm{grad}})$, FDM

That's great ...

... but what about simplices?

Simplicial finite elements for the De Rham complex

Key idea for new elements

Define DOFs as in (Demkowicz et al., 2000) on a reference symmetric simplex Δ^d with a careful choice of polynomials that promote orthogonality in $(\cdot,\cdot)_{L^2}$ and $(\cdot,\cdot)_{H(\mathrm{d})}$.

$$\begin{tikzcd}
H(\mathrm{grad}) \arrow[r, "\mathrm{grad}"] \arrow[d] & H(\mathrm{curl}) \arrow[r, "\mathrm{curl}"] \arrow[d] & H(\mathrm{div}) \arrow[r, "\mathrm{div}"] \arrow[d] & L^2 \arrow[d] \\
\mathrm{CG}_p \arrow[r, "\mathrm{grad}"] & \mathrm{Ned}^1_p \arrow[r, "\mathrm{curl}"] & \mathrm{RT}_p \arrow[r, "\mathrm{div}"] & \mathrm{DG}_{p-1}
\end{tikzcd}$$

Simplicial finite elements for the De Rham complex

Key idea for new elements

Define DOFs as in (Demkowicz et al., 2000) on a reference symmetric simplex Δ^d with a careful choice of polynomials that promote orthogonality in $(\cdot,\cdot)_{L^2}$ and $(\cdot,\cdot)_{H(\mathrm{d})}$.

$$\begin{array}{ccccccc}
H(\mathrm{grad}) & \xrightarrow{\mathrm{grad}} & H(\mathrm{curl}) & \xrightarrow{\mathrm{curl}} & H(\mathrm{div}) & \xrightarrow{\mathrm{div}} & L^2 \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
\mathrm{CG}_p & \xrightarrow{\mathrm{grad}} & \mathrm{Ned}^2_{p-1} & \xrightarrow{\mathrm{curl}} & \mathrm{BDM}_{p-2} & \xrightarrow{\mathrm{div}} & \mathrm{DG}_{p-3}
\end{array}$$

Simplicial FDM degrees of freedom for $H(\mathrm{grad}, \Delta^d)$

Point evaluation at vertices(Δ^d): $\ell_j^V(v) = v(\mathbf{x}_j)$,

Simplicial FDM degrees of freedom for $H(\mathrm{grad}, \Delta^d)$

Point evaluation at vertices(Δ^d): $\ell_j^V(v) = v(\mathbf{x}_j)$,

for each sub-entity $S \in$ edges$(\Delta^d) \cup$ faces$(\Delta^d) \cup$ interior(Δ^d):

$$\ell_j^S(v) = (\mathrm{grad}_S\, \phi_j^S, \mathrm{grad}_S\, v)_S,$$

Simplicial FDM degrees of freedom for $H(\mathrm{grad}, \Delta^d)$

Point evaluation at vertices(Δ^d): $\ell_j^V(v) = v(\mathbf{x}_j)$,

for each sub-entity $S \in$ edges$(\Delta^d) \cup$ faces$(\Delta^d) \cup$ interior(Δ^d):

$$\ell_j^S(v) = (\mathrm{grad}_S\, \phi_j^S, \mathrm{grad}_S\, v)_S,$$

where $\{\phi_j^S\}$ is a basis for $\mathbb{P}_{p,0}(S) = \{v \in \mathbb{P}_p(S) : v = 0 \text{ on } \partial S\}$, s.t.

$$(\mathrm{grad}_S\, \phi_j^S, \mathrm{grad}_S\, \phi_i^S)_S = \delta_{ij}, \quad (\phi_j^S, \phi_i^S)_S = \lambda_j \delta_{ij}.$$

Simplicial FDM degrees of freedom for $H(\mathrm{grad}, \Delta^d)$

Point evaluation at vertices(Δ^d): $\ell_j^V(v) = v(\mathbf{x}_j)$,

for each sub-entity $S \in$ edges(Δ^d) $\cup$ faces(Δ^d) $\cup$ interior(Δ^d):

$$\ell_j^S(v) = (\mathrm{grad}_S\, \phi_j^S, \mathrm{grad}_S\, v)_S,$$

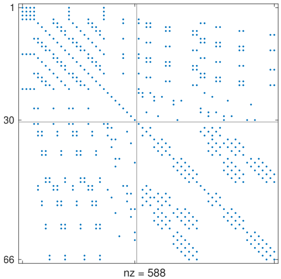
where $\{\phi_j^S\}$ is a basis for $\mathbb{P}_{p,0}(S) = \{v \in \mathbb{P}_p(S) : v = 0 \text{ on } \partial S\}$, s.t.

$$(\mathrm{grad}_S\, \phi_j^S, \mathrm{grad}_S\, \phi_i^S)_S = \delta_{ij}, \quad (\phi_j^S, \phi_i^S)_S = \lambda_j \delta_{ij}.$$

The eigenbases $\{\phi_j^S\}$ are numerically computed offline and only once on the reference interval, triangle, and tetrahedron.

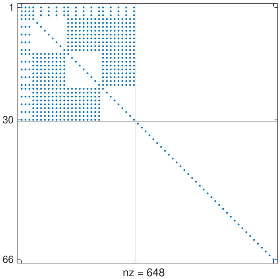
The stiffness and mass matrices A, B have diagonal interior-interior block!
The stiffness matrix does not couple the interior and interface.

A, $H(\mathrm{grad}, \Delta^2)$, Hier.



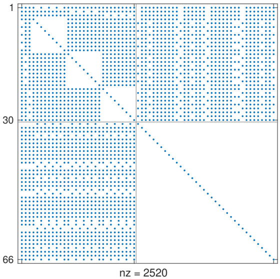
(Beuchler & Schöberl, 2006)

A, $H(\mathrm{grad}, \Delta^2)$, FDM



This work, $p = 10$

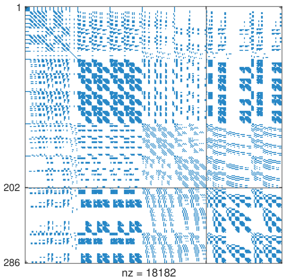
B, $H(\mathrm{grad}, \Delta^2)$, FDM



This work, $p = 10$

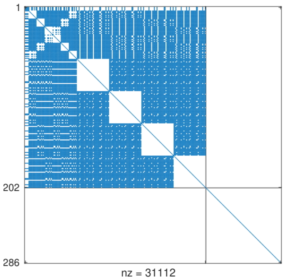
The stiffness and mass matrices A, B have diagonal interior-interior block!
The stiffness matrix does not couple the interior and interface.

A, $H(\mathrm{grad}, \Delta^3)$, Hier.



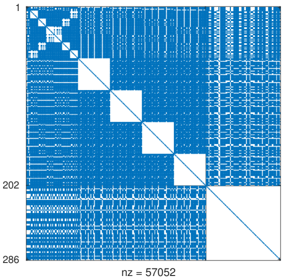
(Beuchler & Pillwein, 2007)

A, $H(\mathrm{grad}, \Delta^3)$, FDM



This work, $p = 10$

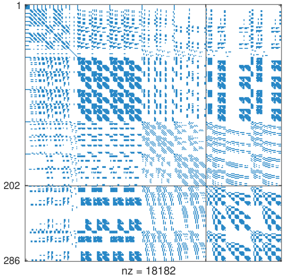
B, $H(\mathrm{grad}, \Delta^3)$, FDM



This work, $p = 10$

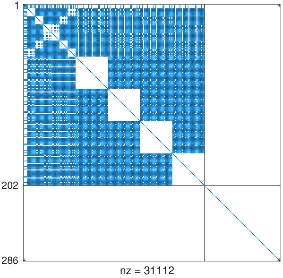
The stiffness and mass matrices A, B have diagonal interior-interior block!
The stiffness matrix does not couple the interior and interface.

A, $H(\mathrm{grad}, \Delta^3)$, Hier.



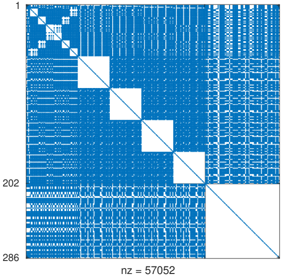
(Beuchler & Pillwein, 2007)

A, $H(\mathrm{grad}, \Delta^3)$, FDM



This work, $p = 10$

B, $H(\mathrm{grad}, \Delta^3)$, FDM

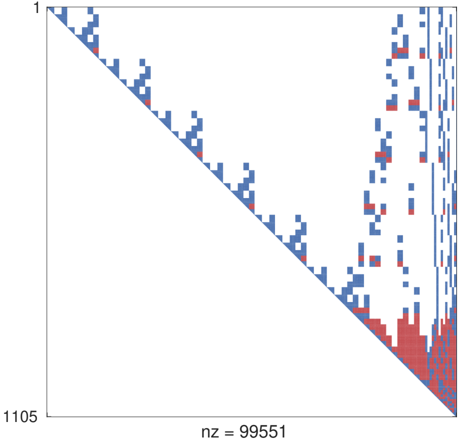


This work, $p = 10$

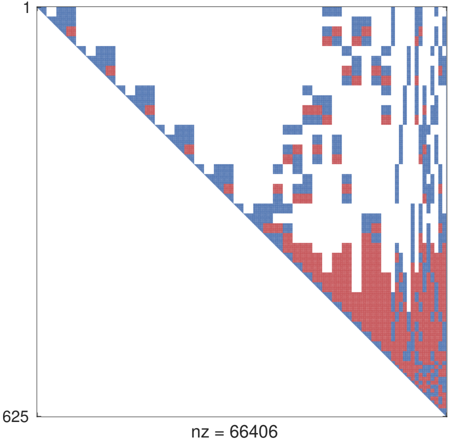
The sparsity does not carry over to generally mapped elements. Our preconditioner discards any coupling between interior DOFs.

The basis functions assemble the Schur complement on Δ^d. We assemble a preconditioner that removes the interior DOFs from the patch problems.

A, Lagrange, 24 cells with interiors



A, FDM, 24 cells without interiors



Simplicial FDM degrees of freedom for $H(\mathrm{curl}, \Delta^d)$

Tangential moments along $E \in$ edges(Δ^d):

$$\ell_j^E(v) = (q_j, v \cdot \tau)_E, \quad q_j \in \mathbb{P}_p(E),$$

Simplicial FDM degrees of freedom for $H(\mathrm{curl}, \Delta^d)$

Tangential moments along $E \in \mathsf{edges}(\Delta^d)$:

$$\ell_j^E(v) = (q_j, v \cdot \tau)_E, \quad q_j \in \mathbb{P}_p(E),$$

and for each sub-entity $S \in \mathsf{faces}(\Delta^d) \cup \mathsf{interior}(\Delta^d)$:

$$\begin{aligned} \ell_j^{S,0}(v) &= (\mathrm{grad}_S\, \phi_j^S, v)_S, \\ \ell_j^{S,1}(v) &= (\mathrm{curl}_S\, \Phi_j^S, \mathrm{curl}_S\, v)_S, \end{aligned}$$

Simplicial FDM degrees of freedom for $H(\mathrm{curl}, \Delta^d)$

Tangential moments along $E \in$ edges(Δ^d):

$$\ell_j^E(v) = (q_j, v \cdot \tau)_E, \quad q_j \in \mathbb{P}_p(E),$$

and for each sub-entity $S \in$ faces$(\Delta^d) \cup$ interior(Δ^d):

$$\begin{aligned} \ell_j^{S,0}(v) &= (\mathrm{grad}_S\, \phi_j^S, v)_S, \\ \ell_j^{S,1}(v) &= (\mathrm{curl}_S\, \Phi_j^S, \mathrm{curl}_S\, v)_S, \end{aligned}$$

where $\{\mathrm{curl}_S\, \Phi_j^S\}$ is a basis for $\mathrm{curl}_S\, \mathbb{X}$, $\mathbb{X} = [\mathbb{P}_p(S)]^d \cap H_0(\mathrm{curl}, S)$, s.t.

$$(\mathrm{curl}_S\, \Phi_j^S, \mathrm{curl}_S\, \Phi_i^S)_S = \delta_{ij}, \quad (\Phi_j^S, \Phi_i^S)_S = \lambda_j \delta_{ij}, \quad \Phi_j^S \times \mathbf{n} = 0 \text{ on } \partial S.$$

Simplicial FDM degrees of freedom for $H(\mathrm{div}, \Delta^d)$

Normal moments on $F \in$ faces(Δ^d):

$$\ell_j^F(v) = (q_j, v \cdot \mathbf{n})_F, \quad q_j \in \mathbb{P}_p(F),$$

and on the interior$(\Delta^d) = K$:

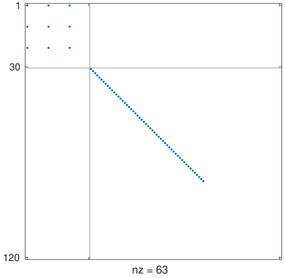
$$\begin{aligned} \ell_j^{K,0}(v) &= (\mathrm{curl}\, \Phi_j^K, v)_K, \\ \ell_j^{K,1}(v) &= (\mathrm{div}\, \Psi_j^K, \mathrm{div}\, v)_K, \end{aligned}$$

where $\{\mathrm{div}\, \Psi_j^K\}$ is a basis for $\mathrm{div}\, \mathbb{Y}$, $\mathbb{Y} = [\mathbb{P}_p(K)]^d \cap H_0(\mathrm{div}, K)$, s.t.

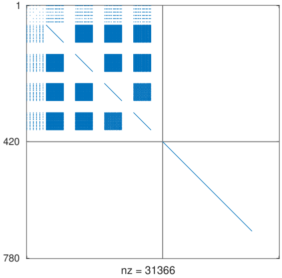
$$(\mathrm{div}\, \Psi_j^K, \mathrm{div}\, \Psi_i^K)_K = \delta_{ij}, \quad (\Psi_j^K, \Psi_i^K)_K = \lambda_j \delta_{ij}, \quad \Psi_j^K \cdot \mathbf{n} = 0 \text{ on } \partial K.$$

The stiffness and mass matrices A, B have diagonal interior-interior block!
The stiffness matrix does not couple the interior and interface.

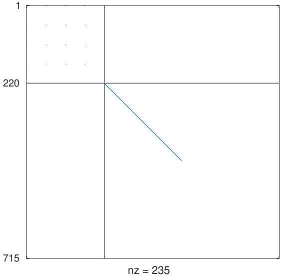
A, $H(\mathrm{curl}, \Delta^2)$, FDM



A, $H(\mathrm{curl}, \Delta^3)$, FDM

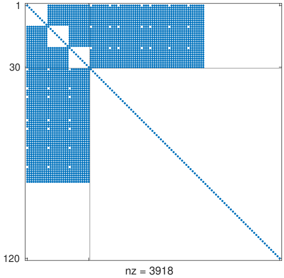


A, $H(\mathrm{div}, \Delta^3)$, FDM

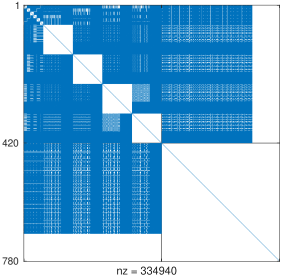


The stiffness and mass matrices A, B have diagonal interior-interior block!
The stiffness matrix does not couple the interior and interface.

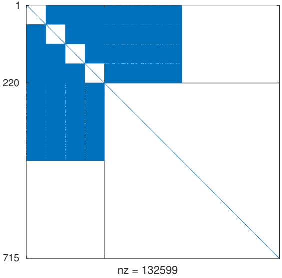
B, $H(\mathrm{curl}, \Delta^2)$, FDM



B, $H(\mathrm{curl}, \Delta^3)$, FDM



B, $H(\mathrm{div}, \Delta^3)$, FDM



Conclusion

- FIAT offers a general framework to construct finite elements.
- We implemented macroelements, enabling higher continuity, divergence-free modes, and tensor symmetry at low polynomial degree.
- We presented simplicial high-order sparsity-promoting bases as a cheaper alternative to statically-condensed patch solvers.