What is libCEED

- C library for performance portable finite element (FE) discretization
  - Write your code once, runs on different hardware
  - Hardware compatability selected @ runtime
- Divided into:

Front-end	Backend
• Where user writes code	• Handles execution of front-end code
• Where physics is	• Diffents per hardware

- Based FE Operator decomposition
- Center for Efficient Exascale Discretizations
- Geared higher-order FE
  - More per DoF (error vs. problem)
  - More computionally efficient (for GPUs)

FE Operator

- Takes solution on FE space and does something with it
  - Represented as coefficients per DoF
- Often represented as a matrix (stiffness, mass, etc.)
  $Ax=b$ type problems ($M\dot{x} + Ax = b$)
- For non-linear problems, we define a residual operator
- G_A is a residual operator
  - it takes solution $\underset{\sim}{V}$, and outputs residual

Example: Laplacian

$$-u_{,ii} = 0 \quad \text{in } \Omega \qquad \Rightarrow \quad -\int_\Omega v u_{,ii} = 0$$
$$u = 0 \quad \text{on } \Gamma$$

IBP:

$$\int_\Omega v_{,i} u_{,i} - \int_\Gamma v u_{,i} \hat{n}_i = 0 \quad \Rightarrow \quad \int_\Omega v_{,i} u_{,i} = 0$$

Choose shape functions to approximate u & v, discard v_B:

$$\int_\Omega N_{B,i} \sum_A N_{A,i} u_A = 0$$

$$\sum_B v_B \int_\Omega N_{B,i} \sum_A N_{A,i} u_A = 0$$

Discretize domain into elements ($\Omega \approx \bigcup_e \Omega^e$) and integrate on parent element:

$$\mathop{\mathbb{A}}_e \int_{\square^e} N^e_{b,i} \xi_{i,j} \left[\sum_a N^e_{a,i} \xi_{i,j} u_a \right] \mathcal{D} = 0$$

Approximate integrals with quadrature:

$$\mathop{\mathbb{A}}_e \sum_q N^e_{b,i}(\xi^q) \xi_{i,j} \xi_{i,j} \left[\sum_a N^e_{a,i}(\xi^q) u_a \right] \mathcal{D} w^q = 0$$

(2) over $N^e_{b,i}(\xi^q)\xi_{i,j}\xi_{i,j}$; (1) over $\left[\sum_a N^e_{a,i}(\xi^q) u_a\right]$; (3) over $N^e_{b,i}(\xi^q)\xi_{i,j}\xi_{i,j}\left[\sum_a N^e_{a,i}(\xi^q) u_a\right]\mathcal{D} w^q$

(0) Move from Global Domain to Local Domain (not shown, $A \to a$, $B \to b$)

(1) Evaluate solution (and derivatives) @ quadrature points

(2) Compute residual @ quad points, Parent → physical corrections, quad weights

(3) Multiply by shape functions (weight function), Sum over quad points

(4) Assemble element matrices Local Domain → Global Domain

Aside Linear Operator (Map)

Mapping from input space to output space such that it is closed by scalar multiplication and vector addition

Given $\mathcal{L}: V \to S$, $\alpha, \beta \in \mathbb{R}$, $v, w \in V$:

$$\mathcal{L}(\alpha v + \beta w) = \alpha \mathcal{L}(v) + \beta \mathcal{L}(w)$$

For finite input + output spaces V^h + S^h,

Any linear operator can be expressed as a matrix

Takeaway: Linear operator $\Leftrightarrow$ Matrix

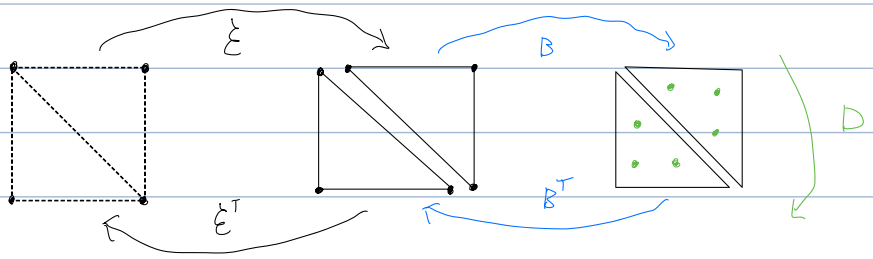
Transpose (adjoint)

Given $\mathcal{Y}: V \to S$, define $\mathcal{Y}^T: S \to V$ such that:

For $v \in V$, $u \in S$ $\quad \langle \mathcal{Y} v, u \rangle = \langle v, \mathcal{Y}^T u \rangle$ ← inner product

Takeaway: Transpose reverses the input + output spaces

FE Operator Decomposition



Given FE Operator A, we can decompose it as:

$$A = \mathcal{E}^T B^T D B \mathcal{E}$$

In action, with solution u_A:

$$A u_A = \mathcal{E}^T B^T D B \mathcal{E} u_A$$

$\mathcal{E}$ Element Restriction

(6×4) matrix size: $\mathcal{E} u_A = u_a$ (4 = u_A, 6 = u_a)

- Maps from Global DoFs → Local DoFs
- Dependent on mesh connectivity
- In matrix form, duplicates shared nodes per element
- $\mathcal{E}^T$ is element assembly

B Basis Evaluator

$(6 \cdot 2 \times 6)$: $B u_a = u_{,i}(\xi^q)$

- Maps Local DoFs → quadrature points
- Dependent on basis choice and quadrature points (locations)
- B^T handles weight function + sum over quadrature points
- B is defined in parent space

D, Quadrature evaluation $(6\cdot2 \times 6\cdot2)$: $Du_{,i}(\xi^a) = R_{,i}(\xi^a)$

- Maps quadrature points to " "
- Evaluates the "action" of the FE operator (@ quadrature points)
- This is where the physics is
- Handles quadrature weights and parent → physical mapping

$$B^T R_{,i}(\xi^a) = R_a$$