4. Automating and Optimizing the Computation of the Element Tensor

Anders Logg
logg@tti-c.org

Toyota Technological Institute at Chicago

Sixth Winter School in Computational Mathematics
Geilo, March 5-10 2006

Automating the computation of the element tensor

Evaluation by quadrature

Evaluation by tensor representation

A language for multilinear forms

Optimizing the computation of the element tensor

Tensor contractions as matrix–vector products

Finding an optimized computation

- Chapters 5 and 7 in lecture notes

The assembly algorithm

$A = 0$
for $K \in \mathcal{T}$
    Compute the element tensor A^K
    (Add A^K to A according to $\{\iota_K\}_{K \in \mathcal{T}}$)
end for

Need to compute the rank r element tensor A^K given by

$$A_i^K = a_K(\phi_{i_1}^{K,1}, \phi_{i_2}^{K,2}, \ldots, \phi_{i_r}^{K,r}) \quad \forall i \in \mathcal{I}_K$$

for any given $K \in \mathcal{T}$

Evaluation by quadrature

- Run-time evaluation by quadrature
- The standard approach
- Basis functions and their derivatives can be pretabulated at the quadrature points
- Used by deal.II and DiffPack

Basic algorithm

$A^K = 0$
for $k = 1, 2, \ldots, N_q$
    for $i \in \mathcal{I}_K$
        $A_i^K += w_k <$ integrand at $x_k >$
    end for
end for

- Quadrature points: $\{x_k\}_{k=1}^{N_q} \subset K$
- Quadrature weights: $\sum_{k=1}^{N_q} w_k = |K|$

Poisson's equation with deal.II

```
...
for (dof_handler.begin_active(); cell! = dof_handler.end(); ++cell)
{
  for (unsigned int i = 0; i < dofs_per_cell; ++i)
    for (unsigned int j = 0; j < dofs_per_cell; ++j)
      for (unsigned int q_point = 0; q_point < n_q_points; ++q_point)
        cell_matrix(i, j) += (fe_values.shape_grad (i, q_point) *
                              fe_values.shape_grad (j, q_point) *
                              fe_values.JxW(q_point));

  for (unsigned int i = 0; i < dofs_per_cell; ++i)
    for (unsigned int q_point = 0; q_point < n_q_points; ++q_point)
      cell_rhs(i) += (fe_values.shape_value (i, q_point) *
                      <value of right-hand side f> *
                      fe_values.JxW(q_point));

  ...
```

Quadrature for Poisson

Bilinear form:

$$a(v, U) = \int_\Omega \nabla v \cdot \nabla U \, \mathrm{d}x$$

Approximate the integral by quadrature:

$$\begin{aligned} A_i^K &= \int_K \nabla \phi_{i_1}^{K,1} \cdot \nabla \phi_{i_2}^{K,2} \, \mathrm{d}x \\ &\approx \sum_{k=1}^{N_q} w_k \nabla \phi_{i_1}^{K,1}(x^k) \cdot \nabla \phi_{i_2}^{K,2}(x^k) \end{aligned}$$

Computing the gradient on K

Compute gradient of ϕ_i^K on K is obtained from the gradient of Φ_i on K_0:

$$\nabla_x \phi_i^K(x_k) = (F_K')^{-\top}(x_k) \nabla_X \Phi_i(X_k)$$

$$\frac{\partial \phi_i^K}{\partial x_j}(x^k) = \sum_{l=1}^{d} \frac{\partial X_l}{\partial x_j}(X^k) \frac{\partial \Phi_i^K}{\partial X_l}(X^k)$$

where $x^k = F_K(X^k)$ and $\phi_i^K = \Phi_i \circ F_K^{-1}$

Cost of quadrature for Poisson

- Computing gradients: $N_q n_0 d^2$ (multiply–add pairs)
- Approximating the integral: $N_q n_0^2 d$
- Total cost for computing the element tensor:

$$N_q n_0 d^2 + N_q n_0^2 d \sim N_q n_0^2 d$$

Also need to compute the mapping F_K, the inverse F_K^{-1} and determinant $\det F_K'$

Evaluation by tensor representation

- Precompute integrals on the reference element K_0
- Used in specialized hand-optimized codes
- Automated by in early versions of DOLFIN for linear elements
- Automated by FFC for general elements
- Integrals automatically precomputed at compile-time

Tensor representation for Poisson

As before we have

$$A_i^K = \int_K \nabla \phi_{i_1}^{K,1} \cdot \nabla \phi_{i_2}^{K,2} \, \mathrm{d}x = \int_K \sum_{\beta=1}^{d} \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\beta} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\beta} \, \mathrm{d}x$$

Make a change of variables:

$$A_i^K = \int_{K_0} \sum_{\beta=1}^{d} \sum_{\alpha_1=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \sum_{\alpha_2=1}^{d} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \det F_K' \, \mathrm{d}X$$

Tensor representation for Poisson

If the mapping F_K is affine, the transforms $\partial X / \partial x$ and the determinant $\det F_K'$ are constant:

$$
\begin{aligned}
A_i^K &= \int_{K_0} \sum_{\beta=1}^{d} \sum_{\alpha_1=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \sum_{\alpha_2=1}^{d} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \det F_K' \, \mathrm{d}X \\
&= \det F_K' \sum_{\alpha_1=1}^{d} \sum_{\alpha_2=1}^{d} \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \int_{K_0} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \, \mathrm{d}X
\end{aligned}
$$

Tensor representation for Poisson

Write as a tensor contraction:

$$A_i^K = \sum_{\alpha_1=1}^{d} \sum_{\alpha_2=1}^{d} A_{i\alpha}^0 G_K^\alpha$$

or

$$A^K = A^0 : G_K$$

where

$$A_{i\alpha}^0 = \int_{K_0} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \, \mathrm{d}X$$

$$G_K^\alpha = \det F_K' \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$$

Cost of tensor representation for Poisson

- Computing the tensor G_K: d^3 (multiply–add pairs)
- Computing the tensor contraction: $n_0^2 d^2$
- Total cost for computing the element tensor:

$$d^3 + n_0^2 d^2 \sim n_0^2 d^2$$

- Compare to cost for quadrature: $N_q n_0^2 d$
- Speedup: N_q / d

The tensor representation $A^K = A^0 : G_K$

The rank r element tensor A^K corresponding to a multlinear form a can be represented as the tensor contraction

$$A^K = A^0 : G_K$$

that is

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A_{i\alpha}^0 G_K^\alpha \quad \forall i \in \mathcal{I}_K$$

- A^0 is the *reference tensor*
- G_K is the *geometry tensor*

Efficient computation of the element tensor

- Compute the reference tensor A^0 once *at compile-time*
- Generate code for efficient computation of the geometry tensor G_K
- Generate code for efficient computation of the tensor contraction $A^K = A^0 : G_K$
- Since A^0 is known at compile-time, we may automatically optimize the tensor-contraction based on the structure of A^0

A canonical form

- Need to find the tensor representation $A^K = A^0 : G_K$ for a general multilinear form
- Write the element tensor in a general canonical form and find the tensor representation for the canonical form:

$$A_i^K = \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^{m} c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j}[\kappa_j(\gamma)] \, \mathrm{d}x$$

The canonical form for Poisson

$$A_i^K = \int_K \sum_{\gamma=1}^{d} \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\gamma} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\gamma} \, \mathrm{d}x = \sum_{\gamma=1}^{d} \int_K \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\gamma} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\gamma} \, \mathrm{d}x$$

$$= \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^{m} c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j} [\kappa_j(\gamma)] \, \mathrm{d}x$$

$$\begin{array}{rcl} m & = & 2 \\ \mathcal{C} & = & [1,d] \\ c(\gamma) & = & (1,1) \end{array} \qquad \begin{array}{rcl} \iota(i,\gamma) & = & (i_1, i_2) \\ \kappa(\gamma) & = & (\emptyset, \emptyset) \\ \delta(\gamma) & = & (\gamma, \gamma) \end{array}$$

The canonical form for a stabilization term

A stabilization term in a least-squares stabilized $\mathrm{cG}(1)\mathrm{cG}(1)$ method for Navier–Stokes:

$$\begin{aligned} a(v,U) &= \int_\Omega (w \cdot \nabla v) \cdot (w \cdot \nabla U) \,\mathrm{d}x \\ &= \int_\Omega \sum_{\gamma_1,\gamma_2,\gamma_3=1}^{d} w[\gamma_2] \frac{\partial v[\gamma_1]}{\partial x_{\gamma_2}} w[\gamma_3] \frac{\partial U[\gamma_1]}{\partial x_{\gamma_3}} \,\mathrm{d}x \end{aligned}$$

where $w \in V_h^3 = V_h^4$ is a given approximation of the velocity

The canonical form for a stabilization term

$$A_i^K = \int_K \sum_{\gamma_1,\gamma_2,\gamma_3=1}^{d} \sum_{\gamma_4=1}^{n_K^3} \sum_{\gamma_5=1}^{n_K^4} \frac{\partial \phi_{i_1}^{K,1}[\gamma_1]}{\partial x_{\gamma_2}} \frac{\partial \phi_{i_2}^{K,2}[\gamma_1]}{\partial x_{\gamma_3}} w_{\gamma_4}^K \phi_{\gamma_4}^{K,3}[\gamma_2] w_{\gamma_5}^K \phi_{\gamma_5}^{K,4}[\gamma_3] \, \mathrm{d}x$$

$$= \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^{m} c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j}[\kappa_j(\gamma)] \, \mathrm{d}x$$

$$\begin{aligned} m &= 4 \\ \mathcal{C} &= [1,d]^3 \times [1,n_K^3] \times [1,n_K^4] \\ c(\gamma) &= (1,1,w_{\gamma_4}^K, w_{\gamma_5}^K) \end{aligned} \qquad \begin{aligned} \iota(i,\gamma) &= (i_1,i_2,\gamma_4,\gamma_5) \\ \kappa(\gamma) &= (\gamma_1,\gamma_1,\gamma_2,\gamma_3) \\ \delta(\gamma) &= (\gamma_2,\gamma_3,\emptyset,\emptyset) \end{aligned}$$

A representation theorem

$$A_i^K = \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^m c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j}[\kappa_j(\gamma)] \, \mathrm{d}x = \sum_{\alpha \in \mathcal{A}} A_{i\alpha}^0 G_K^\alpha$$

where

$$A_{i\alpha}^0 = \sum_{\beta \in \mathcal{B}} \int_{K_0} \prod_{j=1}^m D_X^{\delta_j'(\alpha,\beta)} \Phi_{\iota_j(i,\alpha,\beta)}^j[\kappa_j(\alpha,\beta)] \, \mathrm{d}X$$

and the geometry tensor G_K is the outer product of the coefficients of any weight functions with a tensor that depends only on the Jacobian F_K':

$$G_K^\alpha = \prod_{j=1}^m c_j(\alpha) \, \det F_K' \sum_{\beta \in \mathcal{B}'} \prod_{j'=1}^m \prod_{k=1}^{|\delta_{j'}(\alpha,\beta)|} \frac{\partial X_{\delta_{j'k}'(\alpha,\beta)}}{\partial x_{\delta_{j'k}(\alpha,\beta)}}$$

A representation thereom (cont'd)

- Proof same as for Poisson
- Make a change of variables to the reference cell K_0
- Interchange order of product of summation
- Constructive proof, repeated every time FFC compiles a form

Note that in general, we have a sum of tensor contractions:

$$A^K = \sum_k A^{0,k} : G_{K,k}$$

Test case 1: the mass matrix

Bilinear form:

$$a(v, U) = \int_\Omega v\, U \,\mathrm{d}x$$

Tensor representation:

$$A^0_{i\alpha} = \int_{K_0} \Phi^1_{i_1} \Phi^2_{i_2} \,\mathrm{d}X$$
$$G^\alpha_K = \det F'_K$$

Ranks:

$$|i\alpha| = 2$$
$$|\alpha| = 0$$

Test case 2: Poisson's equation

Bilinear form:

$$a(v, U) = \int_\Omega \nabla v \cdot \nabla U \, \mathrm{d}x$$

Tensor representation:

$$A^0_{i\alpha} = \int_{K_0} \frac{\partial \Phi^1_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi^2_{i_2}}{\partial X_{\alpha_2}} \, \mathrm{d}X$$

$$G^\alpha_K = \det F'_K \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$$

Ranks:

$$|i\alpha| = 4$$

$$|\alpha| = 2$$

Test case 3: nonlinear term in Navier–Stokes

Bilinear form:

$$a(v, U) = \int_\Omega v \cdot (w \cdot \nabla) U \, \mathrm{d}x$$

Tensor representation:

$$A^0_{i\alpha} = \sum_{\beta=1}^{d} \int_{K_0} \Phi^1_{i_1}[\beta] \frac{\partial \Phi^2_{i_2}[\beta]}{\partial X_{\alpha_3}} \Phi^3_{\alpha_1}[\alpha_2] \, \mathrm{d}X$$

$$G^\alpha_K = w^K_{\alpha_1} \det F'_K \frac{\partial X_{\alpha_3}}{\partial x_{\alpha_2}}$$

Ranks:

$$|i\alpha| = 5$$

$$|\alpha| = 3$$

Test case 4: strain-strain term of linear elasticity

Bilinear form:

$$a(v, U) = \int_\Omega \epsilon(v) : \epsilon(U) \, \mathrm{d}x$$

Tensor representation (first term):

$$A^0_{i\alpha} = \sum_{\beta=1}^{d} \int_{K_0} \frac{\partial \Phi^1_{i_1}[\beta]}{\partial X_{\alpha_1}} \frac{\partial \Phi^2_{i_2}[\beta]}{\partial X_{\alpha_2}} \, \mathrm{d}X$$

$$G^\alpha_K = \frac{1}{2} \det F'_K \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$$

Ranks:

$$|i\alpha| = 4$$
$$|\alpha| = 2$$

Extension to non-affine mappings

Consider Poisson again:

$$
\begin{aligned}
A_i^K &= \int_K \nabla \phi_{i_1}^{K,1} \cdot \nabla \phi_{i_2}^{K,2} \, \mathrm{d}x = \int_K \sum_{\beta=1}^{d} \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\beta} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\beta} \, \mathrm{d}x \\
&= \sum_{\alpha_1=1}^{d} \sum_{\alpha_2=1}^{d} \sum_{\beta=1}^{d} \int_{K_0} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \frac{\partial \Phi_{i_1}^{1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^{2}}{\partial X_{\alpha_2}} \det F_K' \, \mathrm{d}X \\
&\approx \sum_{\alpha_1=1}^{d} \sum_{\alpha_2=1}^{d} \sum_{\alpha_3=1}^{N_q} w_{\alpha_3} \frac{\partial \Phi_{i_1}^{1}}{\partial X_{\alpha_1}}(X_{\alpha_3}) \frac{\partial \Phi_{i_2}^{2}}{\partial X_{\alpha_2}}(X_{\alpha_3}) \quad \times \\
&\times \quad \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta}(X_{\alpha_3}) \frac{\partial X_{\alpha_2}}{\partial x_\beta}(X_{\alpha_3}) \det F_K'(X_{\alpha_3})
\end{aligned}
$$

Extension to non-affine mappings

We obtain a representation of the form

$$A^K = A^0 : G_K$$

where the reference tensor A^0 is now given by

$$A^0_{i\alpha} = w_{\alpha_3} \frac{\partial \Phi^1_{i_1}}{\partial X_{\alpha_1}}(X_{\alpha_3}) \frac{\partial \Phi^2_{i_2}}{\partial X_{\alpha_2}}(X_{\alpha_3})$$

and the geometry tensor G_K is given by

$$G^\alpha_K = \det F'_K(X_{\alpha_3}) \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta}(X_{\alpha_3}) \frac{\partial X_{\alpha_2}}{\partial x_\beta}(X_{\alpha_3})$$

A language for multilinear forms

- Language should be close to mathematical notation
- Should be easy to obtain the canonical form from a string in the language
- The tensor representation follows from the canonical form
- Language implemented by FFC

An algebra for multilinear forms

The *vector space* of linear combinations of basis functions:

$$\overline{\mathcal{P}}_K = \left\{ v : v = \sum c_{(\cdot)} \phi_{(\cdot)}^K \right\}$$

The *algebra* of linear combinations of products of basis functions and their derivatives:

$$\overline{\mathcal{P}}_K = \left\{ v : v = \sum c_{(\cdot)} \prod \frac{\partial^{|(\cdot)|} \phi_{(\cdot)}^K}{\partial x_{(\cdot)}} \right\}$$

The elements of the algebra $\overline{\mathcal{P}}_K$ correspond to the integrands of the canonical form

An example

Consider the bilinear form

$$a(v, U) = \int_\Omega \nabla v \cdot \nabla U + v\, U \, \mathrm{d}x$$

with corresponding element tensor canonical form

$$A_i^K = \sum_{\gamma=1}^{d} \int_K \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\gamma} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\gamma} \, \mathrm{d}x + \int_K \phi_{i_1}^{K,1} \phi_{i_2}^{K,2} \, \mathrm{d}x$$

If we let $v = \phi_{i_1}^{K,1} \in \overline{\mathcal{P}}_K$ and $U = \phi_{i_2}^{K,2} \in \overline{\mathcal{P}}_K$ then

$$\nabla v \cdot \nabla U + v\, U \in \overline{\mathcal{P}}_K$$

Implementation by operator-overloading

- Implement by operator-overloading in Python or C++
- Basic types:
  - `BasisFunction`, `Product`, `Sum`
  - `Index`
  - `FiniteElement`
- Basic operators:
  - +, -, $*$ (note absence of /)
  - `D`
  - `dot`, `mult`, `trace`, `transp`
  - `grad`, `div`, `rot`

An example

Bilinear form:

$$a(v, U) = \int_\Omega \nabla v \cdot \nabla U + v\, U \,\mathrm{d}x$$

Implementation:

```
element = FiniteElement(...)

v = BasisFunction(element)
U = BasisFunction(element)

a = (dot(grad(v), grad(U)) + v*U)*dx
```

Computing the tensor contraction

Need to compute the tensor contraction

$$A^K = A^0 : G_K$$

where

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A_{i\alpha}^0 G_K^\alpha$$

for all $i \in \mathcal{I}_K$

Two different ways to efficiently compute the tensor contraction:

- Phrase as matrix–vector products and call BLAS
- Use the structure of the reference tensor to find an optimized computation

The naive algorithm

for $i \in \mathcal{I}_K$
    $A_i^K = 0$
    for $\alpha \in \mathcal{A}$
        $A_i^K = A_i^K + A_{i\alpha}^0 G_K^\alpha$
    end for
end for

- Flatten tensors to vectors and matrices
- Express the tensor contraction $A^K = A^0 : G_K$ as a matrix–vector product

Flattening the tensors

Enumerate $\mathcal{I}_K$ and $\mathcal{A}$:

- Let $\{i^j\}_{j=1}^{|\mathcal{I}_K|}$ be an enumeration the primary multiindices
- Let $\{\alpha^j\}_{j=1}^{|\mathcal{A}|}$ be an enumeration of the secondary multiindices

For Poisson with quadratic elements on triangles, we may take

$$\{i^j\}_{j=1}^{|\mathcal{I}_K|} = \{(1,1),(1,2),\ldots,(1,6),(2,1),\ldots,(6,6)\}$$
$$\{\alpha^j\}_{j=1}^{|\mathcal{A}|} = \{(1,1),(1,2),(2,1),(2,2)\}$$

Flattening the tensors

Define the flattened element and geometry tensors (vectors) $a^K \leftrightarrow A^K$ and $g_K \leftrightarrow G_K$ by

$$a^K = (A^K_{i^1}, A^K_{i^2}, \ldots, A^K_{i^{|\mathcal{I}_K|}})^\top$$
$$g_K = (G_K^{\alpha^1}, G_K^{\alpha^2}, \ldots, G_K^{\alpha^{|\mathcal{A}|}})^\top$$

Define the flattened reference tensor (matrix) $\bar{A}^0$ by

$$\bar{A}^0_{jk} = A^0_{i^j \alpha^k}, \quad j = 1, 2, \ldots, |\mathcal{I}_K|, \quad k = 1, 2, \ldots, |\mathcal{A}|$$

Write as a matrix–vector product

We note that

$$a_j^K = A_{ij}^K = \sum_{\alpha \in \mathcal{A}} A^0_{ij\alpha} G_K^\alpha = \sum_{k=1}^{|\mathcal{A}|} A^0_{ij\alpha^k} G_K^{\alpha^k} = \sum_{k=1}^{|\mathcal{A}|} \bar{A}^0_{jk} (g_K)_k$$

It follows that the tensor contraction

$$A^K = A^0 : G_K$$

corresponds to the matrix–vector product

$$a^K = \bar{A}^0 g_K$$

The general case

In general the element tensor is a sum of tensor contractions:

$$A^K = \sum_k A^{0,k} : G_{K,k}$$

We may still compute the element tensor by a matrix–vector product:

$$a^K = \sum_k \bar{A}^{0,k} g_{K,k} = \begin{bmatrix} \bar{A}^{0,1} & \bar{A}^{0,2} & \cdots \end{bmatrix} \begin{bmatrix} g_{K,1} \\ \\ g_{K,2} \\ \vdots \end{bmatrix} = \bar{A}^0 g_K$$

Processing batches of elements

- Matrix–vector product computed by a level 2 BLAS call
- More efficient to compute a matrix–matrix product with a level 3 BLAS call than a set of matrix–vector products with level 2 BLAS calls

Compute the element tensors for a batch $\{K_k\}_k \subset \mathcal{T}$:

$$\left[a^{K_1} \ a^{K_2} \ \cdots\right] = \left[\bar{A}^0 g_{K_1} \ \bar{A}^0 g_{K_2} \ \ldots\right] = \bar{A}^0 \left[g_{K_1} \ g_{K_2} \ \ldots\right]$$

Finding an optimized computation

- Use the structure of the reference tensor A^0 to find an optimized computation
- The reference tensor may contain zeros
- Take advantage of symmetries
- Look for "complexity-reducing" relations

Will discuss two different symmetry-reducing relations:

- Collinearity
- Closeness in Hamming distance

Collinearity

Note that

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A^0_{i\alpha} G_K^\alpha = a_i^0 \cdot g_K$$

where the vector a_i^0 is defined by

$$a_i^0 = (A^0_{i\alpha^1}, A^0_{i\alpha^2}, \ldots, A^0_{i\alpha^{|\mathcal{A}|}})^\top$$

If a_i^0 and $a_{i'}^0$ are collinear:

$$a_{i'}^0 = \alpha a_i^0$$

for some $\alpha \in \mathbb{R}$ it follows that

$$A_{i'}^K = a_{i'}^0 \cdot g_K = (\alpha a_i^0) \cdot g_K = \alpha A_i^K$$

Collinearity

- If a_i^0 and $a_{i'}^0$ are collinear then $A_{i'}^K$ can be computed from A_i^K in just one multiplication
- Cost of direct computatation is $|\mathcal{A}|$
- For Poisson the cost is reduced from $|\mathcal{A}| = d^2$ to 1
- Look for pairs $(a_i^0, a_{i'}^0)$ that are collinear

Closeness in Hamming distance

- Find pairs $(a_i^0, a_{i'}^0)$ that are close in Hamming distance
- Hamming distance is number of entries that differ
- If the Hamming distance between a_i^0 and $a_{i'}^0$ is ρ then the cost of computing A_i^K from $A_{i'}^K$ is ρ
- Maximum Hamming distance is $\rho = |\mathcal{A}|$

Closeness in Hamming distance

If a_i^0 and $a_{i'}^0$ differ only in the first ρ entries then

$$A_{i'}^K = a_{i'}^0 \cdot g_K = a_i^0 \cdot g_K + \sum_{k=1}^{\rho} (A_{i'\alpha^k}^0 - A_{i\alpha^k}^0) G_K^{\alpha^k}$$
$$= A_i^K + \sum_{k=1}^{\rho} (A_{i'\alpha^k}^0 - A_{i\alpha^k}^0) G_K^{\alpha^k}$$

- The vector $(A_{i'\alpha^1}^0 - A_{i\alpha^1}^0, A_{i'\alpha^2}^0 - A_{i\alpha^2}^0, \ldots, A_{i'\alpha^\rho}^0 - A_{i\alpha^\rho}^0)^\top$ can be computed at compile-time
- The cost is reduced from $|\mathcal{A}|$ to ρ

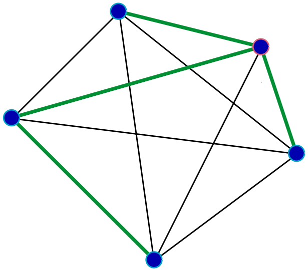
Complexity-reducing relations

- We refer to collinearity and closeness in Hamming distance as *complexity-reducing relations*
- Define the complexity-reducing relation $\rho(a_i^0, a_{i'}^0) \leq |\mathcal{A}|$ as the minimum of all complexity-reducing relations

How do we systematically explore a complexity-reducing relation to find an optimized computation?

Minimum spanning trees (MST)

- Let $G = (V, E)$ be a graph with vertices V and edges E
- A *tree* is any connected acyclic subgraph $G' = (V, E')$ of G with $E' \subset E$
- A *minimum spanning tree* for a weighted graph is a tree with minimal edge weight
- Standard algorithms: Prim's algorith, Kruskal's algorithm



Finding an optimized computation of A^K

- Let the vertices V correspond to the entries $\{A_i^K\}_{i \in \mathcal{I}_K}$ of the element tensor A^K
- Put edges between all pairs of vertices $(A_i^K, A_{i'}^K)$
- Put a weight on each edge $(A_i^K, A_{i'}^K)$ given by $\rho(a_i^0, a_{i'}^0)$
- Compute the minimum spanning tree
- Compute the entries $\{A_i^K\}_{i \in \mathcal{I}_K}$ by traversing the minimum spanning tree

The minimum spanning tree for Poisson

Reference tensor:

$$A^0_{i\alpha} = \int_{K_0} \frac{\partial \Phi^1_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi^2_{i_2}}{\partial X_{\alpha_2}} \, \mathrm{d}X \quad \forall i \in \mathcal{I}_K \quad \forall \alpha \in \mathcal{A}$$

Quadratic elements on triangles:

- $\mathcal{I}_K = [1,6]^2$ and $\mathcal{A} = [1,2]^2$
- The rank two element tensor A^K has 36 entries
- The rank four reference tensor A^0 has 144 entries
- Reduce operation count from 144 to less than 17 multiply-add pairs
- Compute A^K by less than one operation per entry!

The Poisson reference tensor for quadratics on triangles

$(3,3,3,3)$	$(1,0,1,0)$	$(0,1,0,1)$	$(0,0,0,0)$	$-(0,4,0,4)$	$-(4,0,4,0)$
$(1,1,0,0)$	$(3,0,0,0)$	$-(0,1,0,0)$	$(0,4,0,0)$	$(0,0,0,0)$	$-(4,4,0,0)$
$(0,0,1,1)$	$-(0,0,1,0)$	$(0,0,0,3)$	$(0,0,4,0)$	$-(0,0,4,4)$	$(0,0,0,0)$
$(0,0,0,0)$	$(0,0,4,0)$	$(0,4,0,0)$	$(8,4,4,8)$	$-(8,4,4,0)$	$-(0,4,4,8)$
$-(0,0,4,4)$	$(0,0,0,0)$	$-(0,4,0,4)$	$-(8,4,4,0)$	$(8,4,4,8)$	$(0,4,4,0)$
$-(4,4,0,0)$	$-(4,0,4,0)$	$(0,0,0,0)$	$-(0,4,4,8)$	$(0,4,4,0)$	$(8,4,4,8)$

- Scaled by a factor 6
- $A^0_{1111} = A^0_{1112} = A^0_{1121} = A^0_{1122} = 3/6 = 1/2$
- $A^0_{1211} = 1/6,\ A^0_{1212} = 0$ etc.

Use symmetry

- The element tensor A^K is symmetric so we only need to compute 21 of the 36 entries
- The geometry tensor G_K is symmetric so we may reduce the vectors:

$$\begin{aligned} A_i^K &= a_i^0 \cdot g_K = A_{i11}^0 G_K^{11} + A_{i12}^0 G_K^{12} + A_{i21}^0 G_K^{21} + A_{i22}^0 G_K^{22} \\ &= A_{i11}^0 G_K^{11} + (A_{i12}^0 + A_{i21}^0) G_K^{12} + A_{i22}^0 G_K^{22} = \bar{a}_i^0 \cdot \bar{g}_K \end{aligned}$$

where

$$\begin{aligned} \bar{a}_i^0 &= (A_{i11}^0, A_{i12}^0 + A_{i21}^0, A_{i22}^0)^\top \\ \bar{g}_K &= (G_K^{11}, G_K^{12}, G_K^{22})^\top \end{aligned}$$

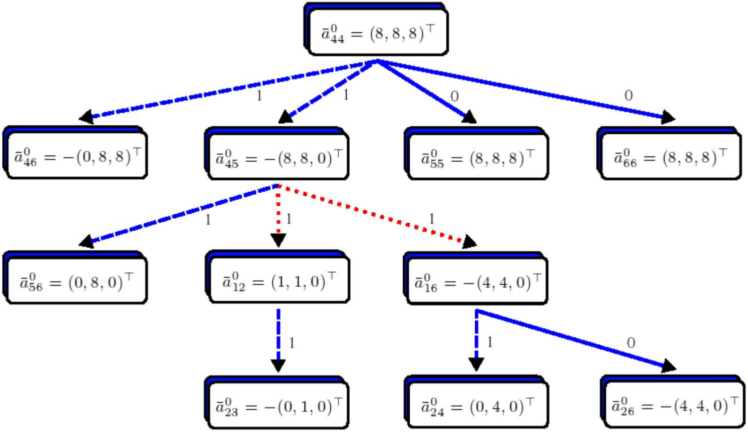
- We directly obtain a reduction from 144 multiply–add pairs to $21 \times 3 = 63$ multiply–add pairs

The symmetry-reduced reference tensor

$(3,6,3)^\top$	$(1,1,0)^\top$	$(0,1,1)^\top$	$(0,0,0)^\top$	$-(0,4,4)^\top$	$-(4,4,0)^\top$
	$(3,0,0)^\top$	$-(0,1,0)^\top,$	$(0,4,0)^\top$	$(0,0,0)^\top$	$-(4,4,0)^\top$
		$(0,0,3)^\top$	$(0,4,0)^\top$	$-(0,4,4)^\top$	$(0,0,0)^\top$
			$(8,8,8)^\top$	$-(8,8,0)^\top$	$-(0,8,8)^\top$
				$(8,8,8)^\top$	$(0,8,0)^\top$
					$(8,8,8)^\top$

- $\bar{a}^0_{12}$, $\bar{a}^0_{16}$, $\bar{a}^0_{26}$ and $\bar{a}^0_{45}$ collinear
- $\bar{a}^0_{44}$ and $\bar{a}^0_{45}$ close in Hamming distance
- Can you spot any other complexity-reducing relations?

The minimum spanning tree (upper part)



The optimized computation

$$A^K_{44} = A^0_{4411}G^{11}_K + (A^0_{4412} + A^0_{4421})G^{12}_K + A^0_{4422}G^{22}_K$$

$A^K_{46} = -A^K_{44} + 8G^{11}_K$	$A^K_{13} = -A^K_{23} + 1G^K_{22}$
$A^K_{45} = -A^K_{44} + 8G^{22}_K$	$A^K_{14} = 0A^K_{23}$
$A^K_{55} = A^K_{44}$	$A^K_{34} = A^K_{24}$
$A^K_{66} = A^K_{44}$	$A^K_{15} = -4A^K_{13}$
$A^K_{56} = -A^K_{45} - 8G^{11}_K$	$A^K_{25} = A^K_{14}$
$A^K_{12} = -\frac{1}{8}A^K_{45}$	$A^K_{22} = A^K_{14} + 3G^K_{11}$
$A^K_{16} = \frac{1}{2}A^K_{45}$	$A^K_{33} = A^K_{14} + 3G^K_{22}$
$A^K_{23} = -A^K_{12} + 1G^{11}_K$	$A^K_{36} = A^K_{14}$
$A^K_{24} = -A^K_{16} - 4G^{11}_K$	$A^K_{35} = A^K_{15}$
$A^K_{26} = A^K_{16}$	$A^K_{11} = A^K_{22} + 6G^K_{12} + 3G^K_{22}$

- 17 multiply-add pairs
- Can be reduced further

Upcoming lectures

0. Automating the Finite Element Method
1. Survey of Current Finite Element Software
2. The Finite Element Method
3. Automating Basis Functions and Assembly
4. Automating and Optimizing the Computation of the Element Tensor
5. FEniCS and the Automation of CMM
6. FEniCS Demo Session