FE-Operator Implementation in C++ using tensor structure

IWR

by

Enes Witwit and Marcel Duerr

Contents

1 Introduction

This is the introduction.

2 Tensor product structure

2.1 2D

Assume we have a solution $u \in V$ in the following form:

$$u(x, y) = \sum_{h=1}^{n^2} \psi_h(x, y) u_h, \ u_h \in \mathbb{R} \ \forall h = 1, \dots, n^2 =: N \tag{2.1}$$

where each ψ_h is a shape function. These shape functions are given by a tensor product of one dimensional polynomials. The transformation given by

$$h = j(n-1) + i, \ i < n \leftrightarrow (i, j) \tag{2.2}$$

gives us an alternative representation of u:

$$u(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} \varphi_i(x) \varphi_j(y) u_{ij}, \ i, j = 1, \dots n \tag{2.3}$$

Let $q = (q_1, \dots, q_n)^T$ be quadrature points of a quadrature rule in 1-D with corresponding weights $w = (w_1, \dots, w_n)^T$. We can get a two-dimensional rule by using the tensor product and obtain points $(\boldsymbol{q}_1, \dots, \boldsymbol{q}_N)$ and weights $(\boldsymbol{w}_1, \dots, \boldsymbol{w}_N)$ with $\boldsymbol{q}_h = (q_i, q_j)$ and $\boldsymbol{w}_h = w_i w_j$.

First, we want to evaluate u at every quadrature point. This can be done by a matrix vector multiplication:

$$\begin{pmatrix} \psi_1(\boldsymbol{q}_1) & \dots & \psi_N(\boldsymbol{q}_1) \\ \vdots & \ddots & \vdots \\ \psi_1(\boldsymbol{q}_N) & \dots & \psi_N(\boldsymbol{q}_N) \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} u(\boldsymbol{q}_1) \\ \vdots \\ u(\boldsymbol{q}_N) \end{pmatrix} \tag{2.4}$$

The matrix can be written as a tensor product of two (in this case even identical) matrices:

$$\mathcal{N}^T \otimes \mathcal{N}^T = \begin{pmatrix} \varphi_1(q_1) & \dots & \varphi_n(q_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(q_n) & \dots & \varphi_n(q_n) \end{pmatrix} \otimes \begin{pmatrix} \varphi_1(q_1) & \dots & \varphi_n(q_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(q_n) & \dots & \varphi_n(q_n) \end{pmatrix} \tag{2.5}$$

Let $(\mathcal{U}_{ij}) \in \mathbb{R}^{n\times n}$ be the matrix filled with the coefficients u_{ij}. Using sum factorization we can multiply these matrices instead of using the formula in (4) which gives us

$$\mathcal{N}^T \mathcal{U} \mathcal{N} = \bar{\mathcal{U}}, \quad \bar{\mathcal{U}}_{ij} = u(q_i, q_j) \tag{2.6}$$

The quadrature weights and the summation of the entries can also be done in a efficient way when the tensor product is exploited. Multiplying $\bar{\mathcal{U}}$ with quadrature weights w from the right will sum each weighted column, multyplying w^T from the left will do the rest. The full operation therefore reads

$$\int\int u(x,y)dx\ dy \approx w^T \mathcal{N}^T \mathcal{U} \mathcal{N} w \tag{2.7}$$

So far, we have only integrated u, but we can also test it with an ansatz function without having to change much. Consider some function $f(x,y) = g(x)h(y)$. Since f is seperable, we only have to change the last step. Instead of only multiplying weigths, we will also multiply with the ansatz function evaluated at the respective quadrature point:

$$\bar{w}^x = (w_1 g(q_1), \dots, w_n g(q_n))^T \qquad \bar{w}^y = (w_1 h(q_1), \dots, w_n h(q_n))^T$$

$$\int\int u(x,y) f(x,y) dx\ dy \approx (\bar{w}^x)^T \mathcal{N}^T \mathcal{U} \mathcal{N} \bar{w}^y \tag{2.8}$$

If we have a whole set of ansatz-functions, which are derived from a tensor product of identical one-dimensional ansatz-functions, we can multiply matrices instead of vectors in the last step. This will complete our vmult-operation. We define

$$W = \begin{pmatrix} w1\varphi_1(q_1) & \dots & w_n\varphi_1(q_n) \\ \vdots & \ddots & \vdots \\ w1\varphi_n(q_1) & \dots & w_n\varphi_n(q_n) \end{pmatrix}$$

Since the same $\mathcal{W}\mathcal{N}^T$ is used on both sides, this operation can be done by only three matrix multiplications. We will use a multiplication function, which is also able to multiply by the transposed. This will save us the cost of actually transposing a matrix and has no drawbacks. Final formula:

$$\mathcal{V} = \mathcal{W}\mathcal{N}^T\mathcal{U}(\mathcal{W}\mathcal{N}^T)^T \tag{2.9}$$

This formula, when slightly modified, can be used for other bilinearforms aswell. Consider $(\nabla u, \nabla v)$.

$$\begin{aligned}
(\nabla u, \nabla v) &= (\partial_x u, \partial_x v) + (\partial_y u, \partial_y v) && (2.10)\\
&= \sum_{i,j} u_{ij}(\varphi_i'(x)\varphi_j(y), \phi'(x)\phi(y)) && (2.11)\\
&+ \sum_{i,j} u_{ij}(\varphi_i(x)\varphi_j'(y), \phi(x)\phi'(y)) && (2.12)
\end{aligned}$$

We will introduce the two matrices $\mathcal{W}'$ and $\mathcal{N}'$. These matrices are similar to the matrices above, but instead of evaluating the ansatz-functions they will evaluate their derivative. The resulting formula is given by

$$\mathcal{V} = \mathcal{W}'\mathcal{N}'^T\mathcal{U}(\mathcal{W}\mathcal{N}^T)^T + \mathcal{W}\mathcal{N}^T\mathcal{U}(\mathcal{W}'\mathcal{N}'^T)^T$$

Consider $(-\Delta u, v) = -(\partial_{xx} u, v) + -(\partial_{yy} u, v)$.
We introduce $\mathcal{N}''$, which evaluates the second derivatives of the ansatz-functions at the quadrature points. Note that the matrix $\mathcal{W}$ is not modified here. As a result we get

$$\mathcal{V} = -\mathcal{W}\mathcal{N}''^T\mathcal{U}(\mathcal{W}\mathcal{N}^T)^T - \mathcal{W}\mathcal{N}^T\mathcal{U}(\mathcal{W}\mathcal{N}''^T)^T$$

2.2 3D

3 Implementation

3.1 2D

3.2 3D