Fast Implementation of mixed finite elements in MATLAB

Teddy Weinberg

mentor: Bedřich Sousedík

Department of Mathematics and Statistics
University of Maryland, Baltimore County

Supported by the National Science Foundation (award DMS-1521563) and by an Undergraduate Research Award from the UMBC Office of Undergraduate Education.

DE seminar, April 30, 2018

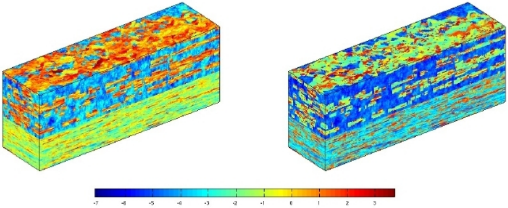
Objectives

- Study partial differential equations modeling flow in porous media.
- Implement vectorized finite element method to simulate flow in porous media in MATLAB for various elements.
- Compare the effectiveness of the efficient implementation with that of the standard approach by numerical experiments.

Flow in Porous media

Understanding flow in porous media is important for many applications

- Managing groundwater reserves
- Maintaining CO_2 storage facilities
- Simulating petroleum reservoirs.



SPE 10 visualization

Model Problem

From Darcy's law, consider the model problem:

$$\mathbb{k}^{-1}\vec{u} + \nabla p = 0, \tag{1}$$

$$\nabla \cdot \vec{u} = f, \tag{2}$$

where

$\mathbb{k}$ ... permeability coefficient,

$\vec{u}$ ... velocity,

p ... pressure,

f ... source terms.

Weak Formulation

In the mixed variational formulation of (1)-(2) we wish to find $(\vec{u}, p) \in (U_E, Q)$ such that

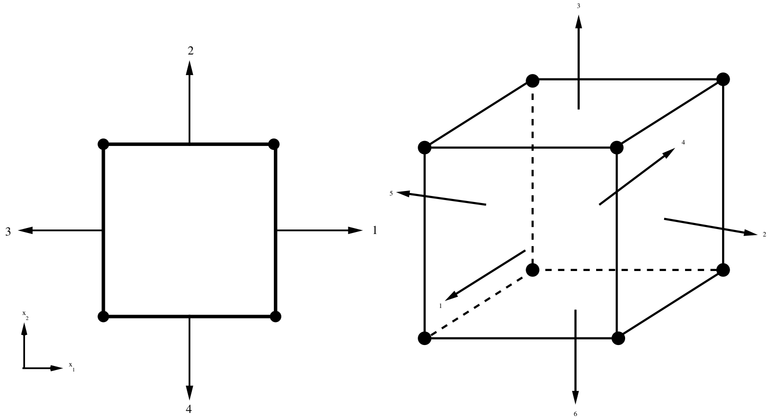
$$\int_\Omega \mathbb{k}^{-1}\vec{u}\cdot\vec{v}\,dx - \int_\Omega p\nabla\cdot\vec{v}\,dx = 0, \qquad \forall\vec{v}\in U, \tag{3}$$

$$-\int_\Omega \nabla\cdot\vec{u}q\,dx = -\int_\Omega fq\,dx, \quad \forall q\in Q, \tag{4}$$

where the pair of spaces (U, Q) is selected so that $U \subset \mathbf{H}(\Omega; \mathrm{div})$ and $Q \subset L^2(\Omega)$, and U_E is an extension of U containing velocities that satisfy the essential boundary condition.

Discretization

- We used lowest order Raviart-Thomas finite elements (RT0).
- We implemented for triangular, quadrilateral, tetrahedral, and hexahedral elements.



Matrix Form

Let the basis functions for the velocity and pressure spaces be denoted φ_i and ψ_j, respectively. In matrix terminology, the discretization of (3)–(4) can be written as a saddle-point linear system

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{f} \end{bmatrix}, \tag{5}$$

where

$$\mathbf{A} = [a_{ij}], \qquad a_{ij} = \int_\Omega \mathbb{k}^{-1}\, \varphi_i \cdot \varphi_j \, dx,$$

$$\mathbf{B} = [b_{k\ell}], \qquad b_{k\ell} = -\int_\Omega \nabla \cdot \varphi_k \, \psi_\ell \, dx,$$

$$\mathbf{f} = [f_\ell], \qquad f_\ell = -\int_\Omega f \, \psi_\ell \, dx$$

Vectorization

- In MATLAB, iterative structures, also known as loops, are not efficient
- However, matrix operations are very efficient
- By converting loops to matrix operations, we can significantly increase the speed of our code.

Vectorization

- We implemented both a standard and a vectorized version of our code.
- The standard version finds the stiffness matrices by looping through each element.
- The vectorized version calculates all of the local element matrices simultaneously.
- The assembly process of the local element matrices into the global system was also vectorized.

Non-Vectorized Code

```
for e = 1:nelem % loop over elements
    A = zeros(nelf);
    B = zeros(1,nelf);
    xcoord = nodes2coord(1,elems2nodes(:,e));
    ycoord = nodes2coord(2,elems2nodes(:,e));
    for intx = 1:nglx
        x = point2(intx,1);
        wtx = weight2(intx,1);
        for inty = 1:ngly
            y = point2(inty,2);
            wty = weight2(inty,2);
            [sqhape,divsqhape,dhdr,dhds] = feisoquad2D4n_RT0(x,y);
            J = [ (xcoord(2)-xcoord(1))/2 0;
                0 (ycoord(4)-ycoord(1))/2 ]; %Jacobian
            detJ = det(J);
            sqhape1 = spdiags(signs(:,e),0,nelf,nelf) * 1/(detJ) * (sqhape*J) * ...
                spdiags(coeffs(:,e),0,dim,dim);
                sqhape2 = spdiags(signs(:,e),0,nelf,nelf) * 1/(detJ) * (sqhape*J);
            A = A + (sqhape1*sqhape2')*wtx*wty*detJ;
            B = B + ( signs(:,e)' .* divsqhape )*wtx*wty*detJ;
        end
        Kloc(:,:,e) = [ A B';
                B 0 ];
    end
end
```

Vectorized Code

```
Kloc = zeros(nbasis+1,nbasis+1,nelems);
for i=1:nip
    for m=1:nbasis
        for k=m:nbasis
            Kloc(m,k,:) = squeeze(Kloc(m,k,:))' + ...
                w(i) .* B_K_detA'.^(-1) .* ...
                sum( squeeze( astam(signs(:,m), ( amsv(B_K, val(i,:,m)) .* ...
                reshape(coeffs,size(coeffs,1),1,size(coeffs,2)) ) ) ).* ...
                squeeze( astam(signs(:,k), amsv(B_K, val(i,:,k))) ) ...
                );
            end
            Kloc(m,nbasis+1,:) = squeeze(Kloc(m,nbasis+1,:)) + ...
                w(i) .* B_K_detA.^(1) .* ...
                (signs(:,m) .* dval(i,:,m) );
        end
    end
Kloc = copy_triu(Kloc);
```

Testing

- Experiments were performed on a (0,1)x(0,1) domain for 2-D and a (0,1)x(0,1)x(0,1).
- The domain was discretized into smaller equally sized squares or blocks used for setup of linear system
- The experiments were run on a computer with two 8-core Intel Xeon E5-2620v4 2.10 GHz procesors with 1 TB memory and Linux openSUSE 42.3.

Results (Quadrilaterals)

problem setup	standard			vectorized		
partitioning	t_e	t_a	$t_e + t_a$	t_e	t_a	$t_e + t_a$
4×4	.13	.01	.14	.06	.02	.08
8×8	.03	.00	.03	.00	.01	.02
16×16	.08	.02	.10	.02	.01	.03
32×32	.03	.10	.38	.02	.03	.05
64×64	1.10	.94	2.04	.06	.15	.21
128×128	5.37	11.96	17.33	.15	.74	.88
256×256	20.02	196.49	216.51	.51	2.25	2.76
512×512	70.12	5163.50	5233.62	2.07	9.55	11.62

Results (Blocks)

problem setup	standard			vectorized		
partitioning	t_e	t_a	$t_e + t_a$	t_e	t_a	$t_e + t_a$
$4 \times 4 \times 4$	.18	.01	.19	.04	.02	.06
$8 \times 8 \times 8$	.31	.09	.40	.03	.02	.05
$16 \times 16 \times 16$	.2.35	2.47	4.82	.15	.14	.29
$32 \times 32 \times 32$	18.73	147.96	166.69	.84	.98	1.82
$64 \times 64 \times 64$	148.95	18822	18970.95	7.42	8.13	15.55

Conclusions

- As expected, the vectorized version significantly outperformed the standard version for large numbers of elements
- The vectorized version had runtime increase approximately linearly with the number of elements

Future Research

- Our next step is to work on efficiently solving the linear system produced by the current code.
- We want to develop and implement preconditioners for iterative solvers.
- We want to determine a posteriori error estimates for our computed solutions.