Finite Element Methods and Vectorized Procedures in MATLAB

Jonathan Fritz

thesis advisor: Bedřich Sousedík

Department of Mathematics and Statistics
University of Maryland, Baltimore County

Supported by National Science Foundation DMS-1521563

Senior thesis presentation, May 9, 2016

Motivation

- The Finite Element Method (FEM) is a popular numerical method for solving partial differential equations
- `Matlab` is suitable for rapid prototyping of numerical algorithms
- However: for-loops are slow in Matlab
- ... **Vectorization**
- Our goal: extension of `MATLAB` code by Rahman, Valdman (2013) from triangular to quadrilateral finite elements

History

- Original work can be traced back to Alexander Hrennikoff and Richard Courant (1940s)
- Mostly an engineering technique in its inception
- Rigorous mathematical basis summarized by I. Babuška and K. Aziz (1972)
- Since then FEM has become wide-spread and well-studied

Overview of FEM in 1D

- How FEM works in a 1-dimensional problem

$$-u'' = f(x), \quad u(0) = u(1) = 0$$

- Define a new function space

$$V := \{v : [0,1] \to \mathbb{R} \mid v \textit{ is continuous}, v(0) = v(1) = 0\}$$

- Turn into a variational problem (the weak form)

$$-\int_0^1 u'' v \, dx = \int_0^1 u' v' \, dx = \int_0^1 fv \, dx, \forall v \in V$$

Overview of FEM in 1D

- Finite dimensional subspace $V_h \subset V$

$$V_h = \{u_h : u_h = \sum_{j=1}^{n} u_j \varphi_j\}$$

- Set of basis functions that span V_h

$$\{\varphi_1, \varphi_2, ..., \varphi_n\}$$

Overview of FEM in 1D

- Discrete form of variational problem

$$\int_0^1 u_h' v_h' \, dx = \int_0^1 f v_h \, dx$$

- This leads to a linear system of equations

$$\sum_{j=1}^{n} a_{ij} u_j = \int_0^1 f \varphi_i \, dx \quad i = 1, \dots, n$$

$$Ku = f$$

Overview of FEM in 2D

- Finite Element Analysis can be extended to solutions of PDEs in higher dimensions

$$-\triangle u = f$$

- Boundary Conditions

$$u = 0 \quad \text{on} \quad \partial\Omega,$$

- Function Spaces

$$L_2(\Omega) = \left\{ v : \int_\Omega v^2 \, dx < \infty \right\}$$

$$H^1(\Omega) = \left\{ v \in L_2(\Omega) : \frac{\partial v}{\partial x_i} \in L_2(\Omega), \; i = 1, ..., d \right\}$$

$$H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \; on \; \partial\Omega \right\}$$

Overview of FEM in 2D

- The weak formulation

$$\int_{\Omega} \nabla u_h \cdot \nabla v_h \, dx = \int f v_h \, dx, \quad \forall v_h \in V_h \subset H_0^1(\Omega),$$

Overview of FEM in 2D

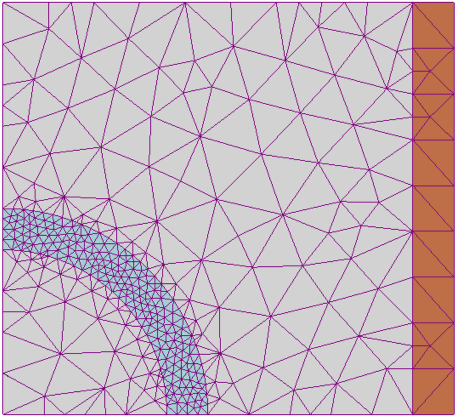

Figure: An example of discretization of a 2-dimensional domain. Image source: Wikipedia

Mapping

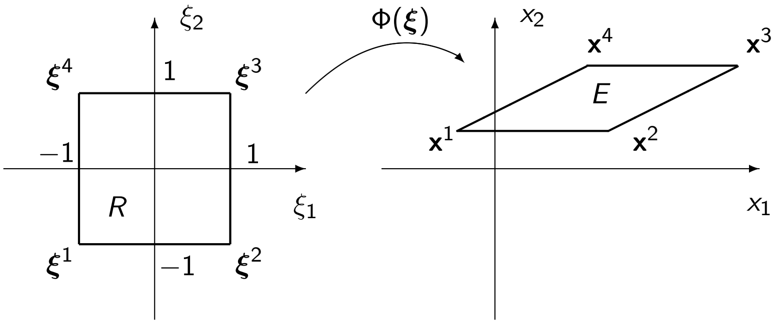

Figure: Finite elements E, R and a mapping $\Phi(\xi)$ between them.

Overview of FEM in 2D

- The domain Ω is discretized into finite elements

$$\mathbf{x} = \Phi\left(\boldsymbol{\xi}\right) = \sum_{a=1}^{4} \varphi_a\left(\boldsymbol{\xi}\right) \mathbf{x}^a, \quad \mathbf{x} = \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right], \quad \boldsymbol{\xi} = \left[\begin{array}{c} \xi_1 \\ \xi_2 \end{array}\right],$$

- The basis functions:

$$\begin{aligned}
\varphi_1\left(\boldsymbol{\xi}\right) &= \tfrac{1}{4}\left(1-\xi_1\right)\left(1-\xi_2\right), \\
\varphi_2\left(\boldsymbol{\xi}\right) &= \tfrac{1}{4}\left(1+\xi_1\right)\left(1-\xi_2\right), \\
\varphi_3\left(\boldsymbol{\xi}\right) &= \tfrac{1}{4}\left(1+\xi_1\right)\left(1+\xi_2\right), \\
\varphi_4\left(\boldsymbol{\xi}\right) &= \tfrac{1}{4}\left(1-\xi_1\right)\left(1+\xi_2\right),
\end{aligned}$$

Overview of FEM in 2D

- The Jacobian matrix of the mapping

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_1}{\partial \xi_2} \\ \frac{\partial x_2}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_2} \end{bmatrix}$$

- From the chain rule

$$\frac{\partial g_a}{\partial x_j} = \frac{\partial}{\partial x_j} \left(\varphi_a \left(\boldsymbol{\xi} \right) \right) = \frac{\partial \varphi_a}{\partial \xi_1} \frac{\partial \xi_1}{\partial x_j} + \frac{\partial \varphi_a}{\partial \xi_2} \frac{\partial \xi_2}{\partial x_j}$$

$$\nabla g_a = \begin{bmatrix} \frac{\partial \xi_1}{\partial x_1} & \frac{\partial \xi_2}{\partial x_1} \\ \frac{\partial \xi_1}{\partial x_2} & \frac{\partial \xi_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial \varphi_a}{\partial \xi_1} \\ \frac{\partial \varphi_a}{\partial \xi_2} \end{bmatrix}$$

Overview of FEM in 2D

- The element stiffness matrix

$$K_{ab}^{E} = \iint\limits_{R} \frac{(J_0 \nabla \varphi_b) \cdot (J_0 \nabla \varphi_a)}{|\det J|} \, d\boldsymbol{\xi}.$$

Numerical Quadrature

$$\iint\limits_R f(x_1, x_2)dx_1dx_2 \approx \sum_{i=1}^{4} w_i f(\mathbf{n}^i)$$

$$\mathbf{n}^i = (\pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}}), \quad w_i = 1,$$

Standard Computation of Element Matrices

```
for e=0 to Number of Elements do
    xcoord, ycoord = Get coordinates of the element nodes;
    matmtx = Get coefficients of the element;
    for intx=1 to Number of Gauss-Legendre Quadrature Points do
        x, wtx = Get sample point, Get Weight;
        for inty=1 to Number of Gauss-Legendre Quadrature Points do
            y, wty = Get Sample Point, Get Weight;
            dhdr, dhds = Get derivatives at quadrature points;
            J, invJ, detJ = Compute the Jacobian matrix (inverse,
            determinant);
            dhdx, dhdy = Get derivatives WRT physical coordinates;
            Kloc = Kloc +
            (dhdx'*matmtx*dhdx+dhdy'*matmtx*dhdy)*wtx*wty*detJ;
        end
    end
end
```

Vectorized Computation of Element Matrices

Vectorization code provided by Rahman and Valdman

```
NE = size(elements,1);
coord = zeros(dim,nlb,NE)
for d = 1:dim
    for i = 1:nlb
        coord(d,i,:) = coordinates(elements(:,i),d);
    end
end
```

Vectorized Computation of Element Matrices

The integration (quadrature) points P

```
IP = [1/3 1/3];
```

The derivatives of the shape functions are provided by

```
[dphi,jac] = phider(coord,IP, P1 );
```

Above 'P1' denotes the integration rule.

Vectorized Computation of Element Matrices

The functions `amtam` and `astam` allow us to work on arrays of matrices

`amtam`

$$C(:,:,i) = A(:,:,i)' * B(:,:,i) \quad \text{for all } i.$$

`astam`

$$C(:,:,i) = a(i) * B(:,:,i) \quad \text{for all } i.$$

Vectorized Computation of Element Matrices

Entries of the finite element matrices
coeffs denote the coefficients

```
Z = astam((areas.*coeffs),amtam(dphi,dphi));
```

Position of these entries in the global stiffness matrix

```
Y = reshape(repmat(elems2nodes,1,nlb),nlb,nlb,nelem);
X = permute(Y,[2 1 3]);
```

The global stiffness matrix K is generated

```
K = sparse(X(:),Y(:),Z(:));
```

This assemby is particularly efficient in MATLAB

Vectorized Computation of Element Matrices

Our contribution: Code extended to quarilateral finite elements

```
function K = setup_SMV(coeffs,nelem,elems2nodes,nodes2coord)

dim = 2;
nlb = 4; % number of local basis functions
coord = zeros(dim,nlb,nelem);
for d = 1:dim
    for i = 1:nlb
        coord(d,i,:) = nodes2coord(d,elems2nodes(i,:));
    end
end
```

Vectorized Computation of Element Matrices

Quadrature rule and derivatives of shape functions

```
p = 1/sqrt(3);
ip = [-p -p; p -p; -p  p; p  p];

[dphi,jac] = phider(coord,ip, Q1 );
jac = abs(jac);
```

Local to global mapping of element entries and setup of stiffness matrix K

```
Y = reshape(repmat(elems2nodes,1,nlb),nlb,nlb,nelem);
X = permute(Y,[2 1 3]);

Z = 0;
for i = 1:size(ip,2)
    dphii = squeeze(dphi(:,:,i,:));
    Z = Z+astam(squeeze(jac(1,i,:)).*coeffs,amtam(dphii,dphii));
end

K = sparse(X(:),Y(:),Z(:));
```

Table: Comparison of the standard and vectorized computation of finite element matrices performed on a ThinkPad Edge laptop. Here *nelem* is the number of elements, *ndof* is the size of the global stiffness matrix K, *nnz* is the number of nonzeros in K, Alg. 1 is the time [s] spent in the standard algorithm, Alg. 2 is the time [s] spent in the vectorized algorithm, and mem is the memory [KB] needed to store K.

nelem	*ndof*	$nnz(K)$	Alg. 1 [s]	Alg. 2 [s]	mem [KB]
4x4	25	169	0.0179	0.0104	2.0820
8x8	81	625	0.0482	0.0103	7.6445
16x16	289	2401	0.1786	0.0142	29.2695
32x32	1089	9409	0.5096	0.0252	114.5195
64x64	4225	37,249	1.9737	0.0620	453.0195
128x128	$16,641$	148,225	8.6355	0.2977	1802.0195
256x256	$66,049$	591,361	66.8047	1.2557	7188.0195

Numerical Experiments

Table: Comparison of the standard and vectorized computation of finite element matrices performed on a MacBook Pro laptop. The headings are same as in Table 1.

nelem	*ndof*	$nnz(K)$	Alg. 1 [s]	Alg. 2 [s]	mem [KB]
4x4	25	169	0.0110	0.0049	2.912
8x8	81	625	0.0187	0.0024	10.656
16x16	289	2401	0.0676	0.0055	40.736
32x32	1089	9409	0.2687	0.0332	159.264
64x64	4225	37,249	1.0745	0.1012	629.792
128x128	16,641	148,225	4.6314	0.1200	2,504.736
256x156	66,049	591,361	25.9865	0.5729	9,990.176
512x512	263,169	2,362,369	233.8331	2.5985	39,903.264

Conclusions

- MATLAB is suited for matrix operations
- For loops iterate over every element, are slow
- Speed can be improved using arrays
- Vectorized Code significantly improves computation speed, however it is also memory intensive