Chapter

A Numerical Simulator Based on Finite Element Method for Diffusion-Advection-Reaction Equation in High Contrast Domains

Hani Akbari

Abstract

Implementation of finite element method (FEM) needs special cares, particularly for essential boundary conditions that have an important effect on symmetry and number of unknowns in the linear systems. Moreover, avoiding numerical integration and using (off-line) calculated element integrals decrease the computational cost significantly. In this chapter we briefly present theoretical topics of FEM. Instead we focus on what is important (and how) to carefully implement FEM for equations that can be the core of a numerical simulator for a diffusion–advection-reaction problem. We consider general 2D and 3D domains, high contrast and heterogeneous diffusion coefficients and generalize the method to nonlinear parabolic equations. Although we use Matlab codes to simplify the explanation of the proposed method, we have implemented it in C++ to reveal the efficiency and examples are presented to admit it.

Keywords: finite element method, diffusion–advection-reaction equation, boundary condition, implementation

1. Introduction

In a domain Ω in $\mathbb{R}^n$, $n = 1, 2, 3,$ consider a partial differential equation of the form of

$$\nabla \cdot (-\kappa \nabla u) + \gamma \cdot \nabla u + \mu u = f \quad \text{in} \quad \Omega, \tag{1}$$

where the unknown function u, reaction coefficient μ and the given function f are scalar functions ($\Omega \to \mathbb{R}$). Diffusion coefficient κ and the (convergence free) advection coefficient γ are defined in Ω and give (normally piecewise constant) values in $\mathbb{R}^{n\times n}$ and $\mathbb{R}^n$, respectively. Eq. (1) is called a diffusion–advection-reaction equation and has immense applications in science and engineering. Transport of heat, momentum and energy, solid mechanics, CO_2 sequestration, computational fluid dynamics and biophysics are a few number of research fields that as a part of the solution strategy need to solve (1) numerically.

After several steps, numerical solution of Eq. (1) is obtained by solving a linear system

$$\mathcal{A}x = b \tag{2}$$

where $\mathcal{A}$ and b are called stiffness matrix and right hand side vector, respectively. In practice each term of diffusion, advection or reaction is accumulated separately in the stiffness matrix.

Several challenges make finding the numerical solution of (1) difficult. 3 of the most important are 1) complex geometry of Ω, 2) heterogeneity and high contrast in coefficient κ that produce very ill-conditioned linear systems and 3) large scale domains. A very common numerical method to solve (1) is FEM that uses a mesh representing the complex geometries accurately and overcomes the first issue. We can employ powerful linear solvers such as multigrid or multiscale methods to resolve the effect of heterogeneity and high contrast in κ in the linear system. Finally a careful implementation in parallel machines reduces computational time of large scale simulations considerably. However, lots of effort and research are still needed to propose a method obtaining a reliable solution in a reasonable time for real problems. Other issues such as uncertainties in the data or nonlinear coupled system of equations should be addressed, as well.

Considering a positive definite κ we explain (and implement) how to obtain a finite element solution of (1) through the following steps.

1. In Matlab we generate arbitrary 2D and simple 3D domains. The main domain is divided into a collection of subdomains, called elements where each element includes some nodes. Numbered nodes and elements generate a mesh. Actually by a mesh we mean 2 data structures, `Cells` (integer valued) that stores identifier or index of nodes in each element and `Nodes` (real valued) that stores coordinates of each node. Many software such as Matlab, GID and Gmesh generate reliable meshes. Corresponding to each element e, a positive definite matrix (a constant or a 2×2 matrix in 2D or a 3×3 matrix in 3D), named κ_e, will be set to form κ. In each element e, we can also set μ_e and γ_e to form μ and γ in Eq. (1), respectively. See Section 2 for details.

2. A brief introduction to FEM is presented that covers weak formulation (Section 3), shape functions and reference elements (Section 4).

3. Evaluating of element integrals and preparing table of calculated integrals are discussed in Section 4.1. Then with help of affine mappings (Section 4.2) we use a linear combination of element integrals to accumulate into the stiffness matrix which is explained in Section 5. Note that different types of elements (for example triangles and rectangles in 2D) might exist in the mesh and we can have an unstructured mesh, generally. Hence element integrals should be considered for any type of elements exist in the mesh.

4. Efficient implementation of boundary conditions, particularly Dirichlet boundary condition is presented in Section 5.2. We show that how a correct but careless implementation of Dirichlet boundary conditions decreases both accuracy and efficiency.

5. After assembly of the linear system (2), we use Matlab functions to solve the linear system and plot the result. Linear solvers are the core of a numerical simulator and their efficiency has a direct impact on the overall simulation. We refer to [1] for further discussions and references.

6. Generalization to more complex equations such as Eq. (25) now becomes straightforward and solving strategy is given in Section 6.

We finish this chapter by introducing the necessary topics to have an independent and efficient simulator in Section 7. Most of the topics presented here are fully discussed in [1–3] which are of great value for further reading.

2. Mesh generation

Implementation of FEM is began with mesh generation which is dividing the main domain into several subdomains or elements. Each element is finite (finite

```
%% Part 1
model = createpde(1);
model.Geometry = multicylinder(2,[1 3 2],'ZOffset',[0 1 4]);
mesh = generateMesh(model,'Hmax',0.5,'GeometricOrder','linear');

figure
subplot 131
pdegplot(model,'FaceLabels','on','CellLabels','on','FaceAlpha',0.25);
subplot 132, pdeplot3D(model);

Nodes = mesh.Nodes;
Cells = mesh.Elements;

Nn = size(Nodes,2); % number of nodes
Nc = size(Cells,2); % number of cells or N_e!
dim = 3;            % dimension

CC2 = findElements(mesh,'region','Cell',2)'; % cell ID of middle region
% for boundary condition:
NFT = findNodes(mesh,'region','Face',6)';    % node ID of top face
NFB = findNodes(mesh,'region','Face',1)';    % node ID of bottom face

%% Part 2
kappa = ones(dim, Nc);
kappa(:,CC2) = 2;

global D Drr Dss Dtt Drs Drt Dst;
load LinearTetraElementIntegral 'D' 'Drr' 'Dss' 'Dtt' 'Drs' 'Drt' 'Dst';

[IM, JM, FFvec, DDvec] = calcMatrices(Nodes, Cells, kappa); % Listing. 3

A = sparse(IM,JM,DDvec+0.1*FFvec,Nn,Nn);

DirV = [ones(length(NFB),1); 10*ones(length(NFT),1)];
DirI = [NFB; NFT];                % Index of nodes with Dirichlet BC
ResI = setdiff((1:Nn)', DirI); % Index of nodes with no Dirichlet BC

b = - A(:,DirI)*DirV;
A(DirI,:) = [];
A(:,DirI) = [];
b(DirI) = [];

sol = A\b;
u = zeros(Nn,1);
u(ResI) = sol;
u(DirI) = DirV;

subplot 133
pdeplot3D(model,'ColorMapData',u);
```

Figure 1.
Creating a geometry and generating its mesh shown in ***Figure*** *2.*

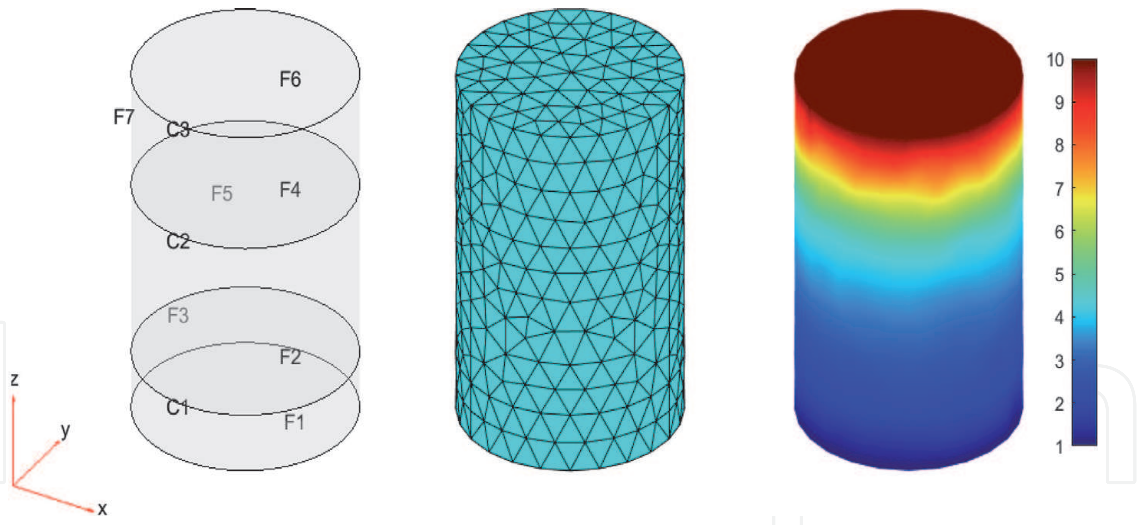

Figure 2.
A 3D domain consists of 3 stacked cylinders (left) and the generated mesh (middle) and the solution of Eq. (21) in right.

elements) and has a regular shape such as triangle or cuboid. Each element has a specific type, such as linear or bilinear, which is defined by shape functions and explained in Section 4. We prefer to use Lagrange triangles and parallelograms in 2D and tetrahedron and cuboid in 3D space as elements. The reason of such selection is to avoid numerical integration to evaluate definite integrals (of Lagrange elements) arise in FEM.

The main output of the mesh generation is two sets, nodes and elements that form the mesh. Nodes are not necessarily the vertices of the elements, for example in linear Crouzeix-Raviart element nodes are placed at edge midpoints or in quadratic Lagrange (triangle) elements midpoints and vertices together form the nodes.

In presence of complex geometries mesh generation can be a challenge for software mesh generators. For not very complicated geometry, Matlab, with a good quality, can generate triangular and tetrahedron mesh for 2D and 3D domains, respectively. We give an example, as shown in Part 1 of **Figure 1**.

First with `createpde (1)` we create a raw model considering one partial differential equation. Then we import or create geometry for this model. For our purposes 3 stacked cylinders with radius 2 and heights 1, 3 and 2 would be fine as shown in **Figure 2**. `pdegplot` shows how Matlab has specified regions (cylinders) and faces by numbering them. Then `generateMesh` generates a mesh with linear (or quadratic by default) tetrahedron elements and can be viewed by `pdeplot3D`. Smaller value for `Hmax` gives a finer mesh. Setting N_n and N_e for number of nodes and elements, respectively, `Nodes` that stores 3 Cartesian coordinates of each node is a matrix of size $3 \times N_n$. Since all elements are linear tetrahedron, `Cells` that stores index of nodes of each element, would be a $4 \times N_e$ matrix. For example the first element of the mesh is formed by 4 nodes, stored in `Cells(:,1)` and `Nodes(:,Cells(:,1))` returns 3D coordinates of that 4 nodes. So we can simply traverse over nodes and elements by their index. Moreover, we can extract element indices of a region with `findElements` or node indices of a region or a face by `findNodes`, where are necessary to set boundary conditions. For example we find node indices of bottom and top faces of the domain in Lines 21–22, since we will set boundary conditions on them.

3. Weak form

In a bounded domain Ω with Lipschitz boundary Γ, Green's formula says for two regular functions u and v we have

$$\int_\Omega \nabla \cdot (-\kappa\nabla u)v = \int_\Omega \kappa\nabla u \cdot \nabla v - \oint_\Gamma \nu \cdot \kappa\nabla u v \tag{3}$$

where ν is the (outward) normal vector. Regularity means that u and v have piecewise continuous (at least) first order partial derivatives to make the above integrals meaningful. So multiplying both side of (1) by v and applying Green's formula we obtain

$$\int_\Omega \kappa\nabla u \cdot \nabla v - \oint_\Gamma \nu \cdot \kappa\nabla u v + \int_\Omega \gamma \cdot \nabla u v + \int_\Omega \mu u v = \int_\Omega f v. \tag{4}$$

Approximation of functions in FEM is done by means of basis functions. Assume $\{\varphi_j\}_j, j = 1, \dots, N_n$ is a set of basis functions, introduced in Section 4, that we can write

$$u = \sum_{j=1}^{N_n} u_j \varphi_j, \tag{5}$$

and our goal is to find unknown coefficients $\{u_j\}_j$. Substituting (5) in (4) and setting $v = \varphi_i, i = 1, \cdots, N_n,$ which is called standard Galerkin method we obtain

$$\sum_{j=1}^{N_n} u_j \int_\Omega \kappa\nabla\varphi_j \cdot \nabla\varphi_i + \sum_{j=1}^{N_n} u_j \int_\Omega \gamma \cdot \nabla\varphi_j\varphi_i + \sum_{j=1}^{N_n} u_j \int_\Omega \mu\varphi_j\varphi_i + \oint_\Gamma \nu \cdot (-\kappa\nabla u)\varphi_i = \int_\Omega f\varphi_i \tag{6}$$

Eq. (6) consists of evaluation of (from left to right) diffusion, advection and reaction terms, boundary integral and right hand side that we calculate them in elements and then sum them up $\left(\int_\Omega = \sum_e \int_e\right)$ to assemble the linear system (2). u or its gradient in boundary integral are known, hence we do not write its expansion form.

4. Shape functions

In practice definite integrals of basis functions and their derivatives in (6) are evaluated in an element, hence restriction of basis functions over elements, called shape functions, contribute in computations. With help of affine mapping we map a typical element into a fixed element, called reference element and evaluate integrals in it. So only shape functions in reference element determine values of integrals in (6).

We use Lagrange elements which means corresponding to each node of an element a shape function is defined such that it is a polynomial in the element, takes value 1 at that specific node and 0 at other nodes of the element. Therefore a basis function in (5) becomes a continues function with value 1 at a specific node and 0 at neighboring nodes with trivial (zero) extension in the rest of the domain, as plotted in **Figure 3**. Note that the derivatives of Lagrange basis functions are not continues on the boundary of the elements.

Some examples of reference elements and their shape functions are presented in **Figure 4**. We see that in linear triangle element where nodes are the vertices of the triangle, the first shape function $S_1(r,s) = 1 - r - s$ corresponding to the first node $n_1 = (0,0)$ has value 1 at n_1 and 0 at $n_2 = (1,0)$ and $n_3 = (0,1)$. It is called a linear element since shape functions are combination of first order polynomials $\{1, r, s\}$.

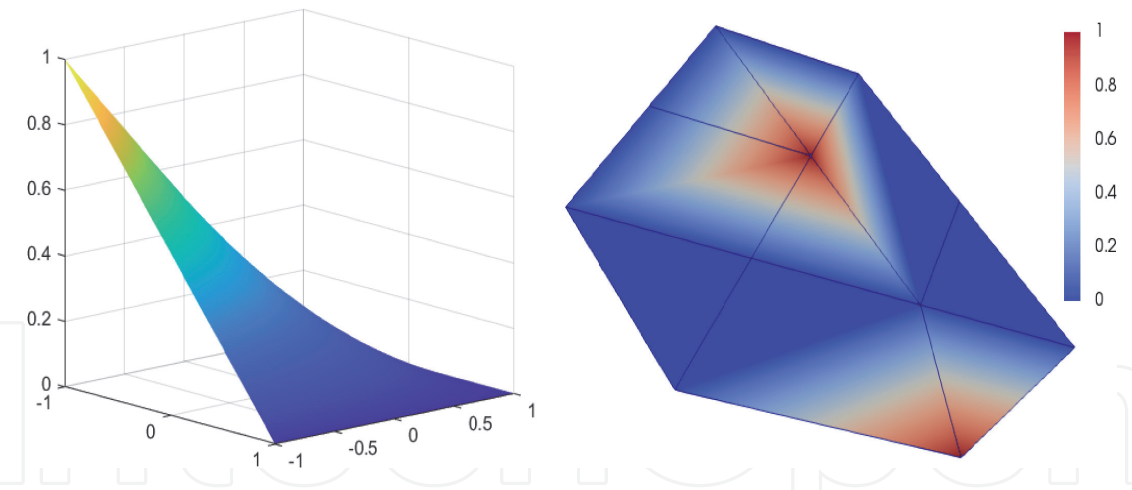

Figure 3.
Left) a shape function of reference bilinear quadrilateral. Right) 2 basis functions where their restrictions in a triangle is linear and in a quadrilateral is bilinear.

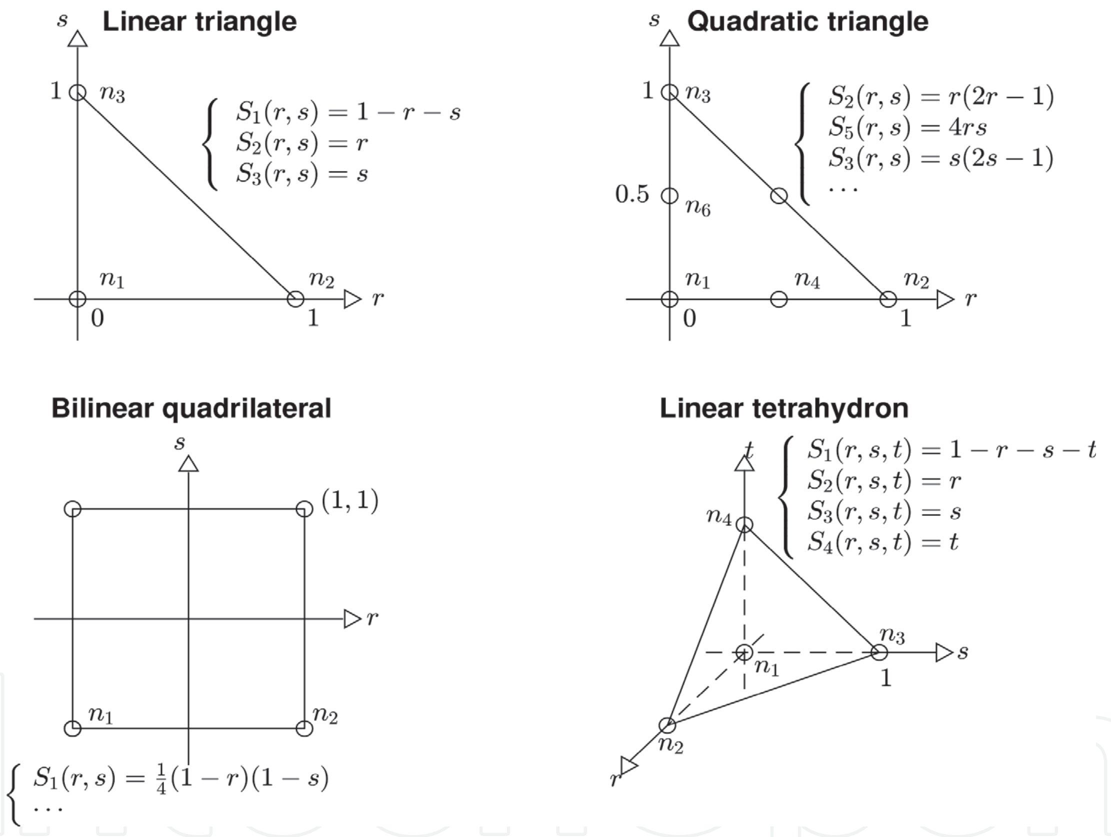

Figure 4.
Some of reference Lagrange elements and their shape functions.

Starting from n_1 nodes are numbered counterclockwise and we consider it for all Lagrange elements.

For Lagrange elements it is easy to find shape functions. For example to find S_1 for bilinear quadrilateral element where nodes are vertices of the square, it suffices to consider the line passing n_2 and n_3 which is $1 - r$ multiplied by the line passing n_3 and n_4 which is $1 - s$. So S_1 has to be of the form $\alpha(1 - r)(1 - s)$ which gives 0 for all nodes except $n_1 = (-1, -1)$. Substituting n_1 in equation of S_1, we can find α such that it gives 1 for n_1. This element is called bilinear since its shape functions are linear combination of $\{1, r, s, rs\}$.

For quadratic triangle element where vertices and midpoints form nodes, for shape function S_1 corresponding to n_1, we need the multiplication of two lines passing n_2 and n_6 which is $0.5 - r - s$ and n_3, n_4 and n_5 which is $1 - r - s$. So S_1 has to be of the form $\alpha(0.5 - r - s)(1 - r - s)$. Substituting n_1 in equation of S_1, we can find α such that it gives 1 for n_1. This element is called quadratic since its shape functions are linear combination of $\{1, r, s, rs, r^2, s^2\}$. Note that we first numbered nodes at vertices and then at edge midpoints.

Exercise 1 Find shape functions of linear tetrahedron as presented in **Figure 4**.

Exercise 2 Quadratic tetrahedron element has nodes at midpoint of edges of linear tetrahedron element, so it has 6 other nodes (numbered 5 to 10) in addition to vertices (numbered 1 to 4 similar to linear tetrahedron). Find the 10 shape functions. For example for node at origin we have $S_1 = 2(1 - r - s - t)(0.5 - r - s - t)$ or for node at $(0, 0, 1)$ we have $S_4 = 2t(t - 0.5)$.

4.1 Element integrals

Definite integrals of shape functions and their derivatives in reference elements, called element integrals, play an important role in assembly of stiffness matrix. Denoting the reference element by $\hat{e}$ which has $N_{\hat{e}}$ nodes and partial derivatives of shape functions by $\partial_r S$ which means $\frac{\partial S}{\partial r}$ we define the matrix D_{rr} (for diffusion term) such that its ijth entry is

$$D_{rr}^{ij} = \int_{\hat{e}} \partial_r S_i \;\; \partial_r S_j, \quad i, j = 1, \cdots, N_{\hat{e}}. \tag{7}$$

Similarly we define D_{rs} and D_{ss}. In 3D space we also have to define D_{rt}, D_{st} and D_{tt}. In **Figure 5** we show how to evaluate D_{rs} for linear tetrahedron element. For example D_{rs} and D_{st} for linear tetrahedron are

$$D_{rs} = \frac{1}{6}\begin{pmatrix} 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad D_{st} = \frac{1}{6}\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Exercise 3 For quadratic tetrahedron element find D_{rs} as

$$D_{rs} = \frac{1}{30}\begin{pmatrix} 3 & 0 & 1 & 0 & -1 & -4 & -1 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & -3 & 0 & 1 & 3 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 4 & 4 & 0 & -4 & 0 & 0 \\ -1 & 0 & -3 & 0 & 4 & 4 & 4 & -4 & 0 & -4 \\ -1 & 0 & 1 & 0 & 4 & 0 & 8 & -4 & 0 & -8 \\ 1 & 0 & 3 & 0 & -4 & -4 & -4 & 4 & 0 & 4 \\ 1 & 0 & -1 & 0 & -4 & 0 & -8 & 4 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Other element integrals such as D_r (for advection term) and D (for reaction or right hand side terms) can be defined where their ijth entries are

$$D_r^{ij} = \int_{\hat{e}} \partial_r S_i \;\; S_j, \quad D^{ij} = \int_{\hat{e}} S_i \;\; S_j, \quad i, j = 1, \cdots, N_{\hat{e}}. \tag{8}$$

```
syms r s t;
Nn = 4;           % number of nodes of reference element
S1 = 1-r-s; S2 = r;  S3 = s;  S4 = t; % shape functions
S = [S1 S2 S3 S4];
Dxy = zeros(Nn,Nn);
for i=1:Nn
   for j=1:Nn
      Drs(i,j) = int(int(int(diff(S(i),r)*diff(S(j),s),t,...
                            [0 1-r-s]), s, [0 1-r]), r, [0 1]);
   end
end
```

Figure 5.
Element integrals D$_{rs}$ *for linear tetrahedron.*

Note that D_{rr} and D are symmetric matrices while D_r is not. Moreover, since $D_{rs} = D_{sr}$ we calculate only one of them. All element integrals are evaluated once and saved for future use. In **Figure 1** we load element integrals of linear tetrahedron that we have calculated and saved in `LinearTetraElementIntegral.mat`.

4.2 Affine mapping on reference elements

So far we introduced shape functions and calculated element integrals, all in reference element. Now we explain how to map an arbitrary element in physical space into the reference element and evaluate integrals in (6) only in terms of element integrals and without any numerical integration. Clearly change of variable is necessary to evaluate integrals when we use a mapping which is explained in next subsection.

Consider an arbitrary triangle, say e, in 2D space with vertices $v_i = (x_i, y_i)$, $i = 1, 2, 3,$ and the reference linear triangle, $\hat{e}$. Set

$$T = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}, \quad \eta = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \xi = \begin{pmatrix} r \\ s \end{pmatrix}, \quad \eta_o = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \tag{9}$$

and define the affine (composition of a linear mapping and a translation) mapping $\hat{e} \to e$ with the rule

$$\xi \to \eta = T\xi + \eta_o. \tag{10}$$

We see that the affine mapping (10) maps the nodes of reference linear triangle, $\{n_i\}_{i=1,2,3}$, to the vertices of the triangle,

$$v_i = Tn_i + v_1, \quad i = 1, 2, 3. \tag{11}$$

Exercise 4 Show that the affine mapping in (9) and (10) also maps the reference quadratic triangle to an arbitrary triangle with 6 nodes (3 at vertices and 3 at edges midpoint).

Exercise 5 Show that the affine mapping,

$$\xi = \begin{pmatrix} r \\ s \end{pmatrix} \to \eta = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2}\begin{pmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{pmatrix}\begin{pmatrix} r \\ s \end{pmatrix} + \frac{1}{2}\begin{pmatrix} x_1 + x_3 \\ y_1 + y_3 \end{pmatrix} \tag{12}$$

maps the reference bilinear quadrilateral to an arbitrary parallelogram. Remember in a parallelogram where v_1 and v_3 are opposite vertices, we have $x_1 + x_3 = x_2 + x_4$ and $y_1 + y_3 = y_2 + y_4$.

Exercise 6 Show that the affine mapping,

$$\xi = \begin{pmatrix} r \\ s \\ t \end{pmatrix} \to \eta = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{pmatrix} \begin{pmatrix} r \\ s \\ t \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{13}$$

maps the reference linear and quadratic tetrahedron to an arbitrary tetrahedron.

4.3 Change of variables in integrals

Consider a scalar function (a typical basis function) $\varphi = \varphi(\eta)$, where $\eta = (x, y)$ in 2D space. Gradient of φ is $\nabla_\eta \varphi = \left(\partial_x \varphi, \partial_y \varphi\right)^t$, where t denotes the transpose of the vector. The chain rule and applying the affine mapping (10) on φ give

$$\nabla_\eta \varphi = T^{-t} \ \nabla_\xi \hat{\varphi} \tag{14}$$

where $\hat{\varphi}(\xi) = \varphi(\eta)$ and T^{-t} is the transpose of the inverse of T. Note that if φ is the restriction of a basis function in element e, then $\hat{\varphi}$ would be the corresponding shape function in $\hat{e}$. Recalling $d\eta = |T| d\xi$, we can write

$$\int_e \kappa_e \nabla_\eta \varphi_i \cdot \nabla_\eta \varphi_j d\eta = \int_e \left(\kappa_e \nabla_\eta \varphi_i\right)^t \nabla_\eta \varphi_j d\eta = \int_{\hat{e}} (\nabla_\xi S_i)^t T^{-1} \kappa_e^t T^{-t} \nabla_\xi S_j \ |T| d\xi. \tag{15}$$

Setting matrix $M_e = |T| T^{-1} \kappa_e^t T^{-t}$ and dropping subscript e as well as η and ξ from ∇, then we have

$$\begin{aligned} \int_e \kappa \nabla \varphi_i \cdot \nabla \varphi_j d\eta = & M_{11} \int_{\hat{e}} \partial_r S_i \ \partial_r S_j \ d\xi \\ & + (M_{12} + M_{21}) \int_{\hat{e}} \partial_r S_i \ \partial_s S_j \ d\xi + M_{22} \int_{\hat{e}} \partial_s S_i \ \partial_s S_j \ d\xi \\ & = M_{11} D_{rr} + (M_{12} + M_{21}) D_{rs} + M_{22} D_{ss} \end{aligned} \tag{16}$$

Exercise 7 Show that in 3D space,

$$\begin{aligned} \int_e \kappa \nabla \varphi_i \cdot \nabla \varphi_j d\eta = & M_{11} D_{rr} + (M_{12} + M_{21}) D_{rs} + (M_{13} + M_{31}) D_{rt} \\ & + M_{22} D_{ss} + (M_{23} + M_{32}) D_{st} \\ & + M_{33} D_{tt}, \end{aligned} \tag{17}$$

where $M = |T| T^{-1} \kappa^t T^{-t}$.
Exercise 8 Referring to notation (8) show that

1. $$\int_e \varphi_i \ \varphi_j d\eta = |T| D \tag{18}$$

2. for a vector γ, defined for each element e in 3D space

$$\int_e \gamma \cdot \nabla \varphi_i \ \varphi_j d\eta = M_1 D_r + M_2 D_s + M_3 D_t \tag{19}$$

where vector $M = |T| T^{-1} \gamma$.

5. Assembly of the linear system

With help of element integrals, affine mapping and Eqs. (17)–(19) we can accumulate each term of (6) into the stiffness matrix of the linear system (2).

5.1 Diffusion term

Equation

$$\int_\Omega \kappa \nabla\varphi_j \cdot \nabla\varphi_i = \sum_e \int_e \kappa_e \nabla\varphi_j \cdot \nabla\varphi_i \tag{20}$$

suggests how to accumulate $\int_\Omega \kappa\nabla u \cdot \nabla v$ into the stiffness matrix:

1. traverse elements and in each element map $\int_e \kappa_e \nabla\varphi_i \cdot \nabla\varphi_j$ into the reference element

2. with help of element integrals and Eq. (17) evaluate the desired term

3. map the local matrix into the stiffness matrix.

Exercise 9 Follow the above steps to accumulate advection and reaction terms into the stiffness matrix.

In **Figure 6** we implemented accumulation of diffusion and reaction terms for linear or quadratic tetrahedron. Note that other elements only have a different affine mapping T and dimension, if we had elements in 2D space. We assume that κ is a diagonal matrix for each element, so a matrix storage of size $3 \times N_c$ can store κ of all elements. We also assume that reaction coefficient μ is constant, otherwise a vector of size N_c can store μ for all elements and should be an input argument. The evaluated integrals are saved such that the indices of row and column and value of the integral are set in `IM`, `JM` and `DDvec` (for diffusion term), respectively. So Line 33 of **Figure 1** is to assembly the stiffness matrix of the equation

$$\int_\Omega \kappa\nabla u \cdot \nabla v + 0.1\int_\Omega uv = 0 \tag{21}$$

where κ is set in Lines 25–26. We also set boundary conditions at bottom and top faces of the boundary (with values 1 and 10, respectively) and explain how to implement it in next subsection to obtain the solution of the problem, plotted in **Figure 2**.

5.2 Boundary integral

Probably correct implementation of boundary integrals is the most important part of the FEM, since boundary conditions mainly determine the situation of physical problem. Neumann, Robin and Dirichlet are 3 types of boundary conditions that might be considered on different parts of the boundary. Although boundary conditions are set on the boundary, they only affect on boundary nodes.

Neumann boundary condition. $\nu \cdot (-\kappa\nabla u) = g$ is a Neumann boundary condition, so boundary integral in (6) becomes $g\oint \varphi_i$. If g is zero, then nothing has to be done for the boundary integrals. In fact incorporating only diffusion–advection-

```
function [IM, JM, FFvec, DDvec] = calcMatrices(Nodes, Cells, kappa)

% Nn = number of nodes ( Nn = size(Nodes,2)), Ne = number of elements.
% Then A1 = sparse(IM,JM,DDvec,Nn,Nn) and A2 = sparse(IM,JM,FFvec,Nn,Nn)
% assembly the diffusion and reaction terms, respectively.
% kappa is considered a diagonal matrix for each element (3*Nc)

  global D Drr Drs Drt Dss Dst Dtt; % saved element integrals

  % ln is the number of nodes of each cell (4 for linear tetrahedron)
  ln = size(Cells,1);
  ln2 = ln*ln;
  ne = size(Cells,2); % number of elements
  NN = ln2*ne;

  IM = zeros(NN,1);
  JM = zeros(NN,1);
  FFvec = zeros(NN,1);
  DDvec = zeros(NN,1);

  lk = 0;
  for K=1:ne    % traverse elements

        enodes = Cells(:,K); % (global) index of the nodes of element K
        x = Nodes(1,enodes); % x-coordinate of the nodes of element K
        y = Nodes(2,enodes); % y-coordinate
        z = Nodes(3,enodes); % z-coordinate

        T = [[x(2)-x(1) x(3)-x(1) x(4)-x(1)];
             [y(2)-y(1) y(3)-y(1) y(4)-y(1)];
             [z(2)-z(1) z(3)-z(1) z(4)-z(1)]];

        detT = det(T);
        TTinv = inv(T)';
        M = detT*(TTinv')*diag(kappa(:,K))*TTinv;

        locDD = M(1,1)*Drr + (M(1,2) + M(2,1))*Drs + (M(1,3)+M(3,1))*Drt + ...
                M(2,2)*Dss + (M(2,3) + M(3,2))*Dst + ...
                M(3,3)*Dtt;

        locFF = detT*D;

        % use indices of nodes to map local matrices to global index
        locI = repmat(enodes,1,4);
        IM(lk+1:lk+ln2) = reshape(locI',[],1);
        JM(lk+1:lk+ln2) = locI(:);
        DDvec(lk+1:lk+ln2) = locDD(:);
        FFvec(lk+1:lk+ln2) = locFF(:);

        lk = lk + ln2;
  end % end for
end % end function
```

Figure 6.
Assembly of diffusion–reaction terms for (linear or quadratic) tetrahedron.

reaction terms and right hand side function means that we have considered a pure Neumann problem and this is exactly what we do in practice. After that we add requested boundary conditions to the linear system of (2). For example if g is non-zero on part of the boundary, say Γ_N, then $-g\oint\varphi_i$ is known for indices that lie on Γ_N and should be added to the ith entry of b in (2).

Dirichlet boundary condition. $u = g$ is a Dirichlet boundary condition, which means value of u at some nodes is known. So if we decompose node indices into 2 sets, say U and K, then we can write

$$u = \sum_{j \in U} u_j \varphi_j + \sum_{j \in K} u_j \varphi_j, \tag{22}$$

where U and K include indices of unknown and known values of u, respectively. If with permutation vector $[U, K]$ we reorder rows and columns of the linear system in (2), then we can write

$$\mathcal{A}u = \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ \mathcal{A}_{21} & \mathcal{A}_{22} \end{pmatrix} \begin{pmatrix} u_U \\ u_K \end{pmatrix} = \begin{pmatrix} b_U \\ b_K \end{pmatrix} \tag{23}$$

The first line gives a smaller linear system

$$\mathcal{A}_{11} u_U = b_U - \mathcal{A}_{12} u_K \tag{24}$$

which preserves the symmetry or positive definiteness of the original problem. In Lines 35–42 of **Figure 1** we showed how to implement it in Matlab. Setting values 1 and 10 for bottom and top faces, we expect a smooth solution starting from 1 at the bottom and reaching to 10 at the top as shown in **Figure 2**.

The second approach to set Dirichlet condition that does not use permutation of the linear system is setting $\mathcal{A}_{21}$ and $\mathcal{A}_{22}$ in (23) to zero and identity matrices, receptively and solving the modified linear system. Although this approach gives the correct solution theoretically, it is very error-prone numerically as shown in a test case in **Figure 7** and explained as follows.

A domain with cuboid elements is refined in the middle and along the x-axis to simulate a fracture in the domain. We solve $\nabla \cdot (-\kappa \nabla u) = 0$ where κ in fracture is 100 times larger than rest of the domain. Setting 10^6 and 5×10^6 as Dirichlet boundary condition on left and right faces (along x-axis) of the domain, respectively, a correct solution should be started from 10^6 and monotonically reached to 5×10^6. The linear system is symmetric positive definite and preconditioned conjugate gradient method is employed to solve it. The correct solution shown in left image of **Figure 7** is obtained by (24). However, the second approach gives the nonphysical (wrong) solution along the fracture as shown in right. Moreover, the correct solution is obtained 3 times faster than the wrong one, mainly because the second approach violates the symmetry (but preserves the positive definiteness) of the linear system, due to setting $A_{21} = 0$.

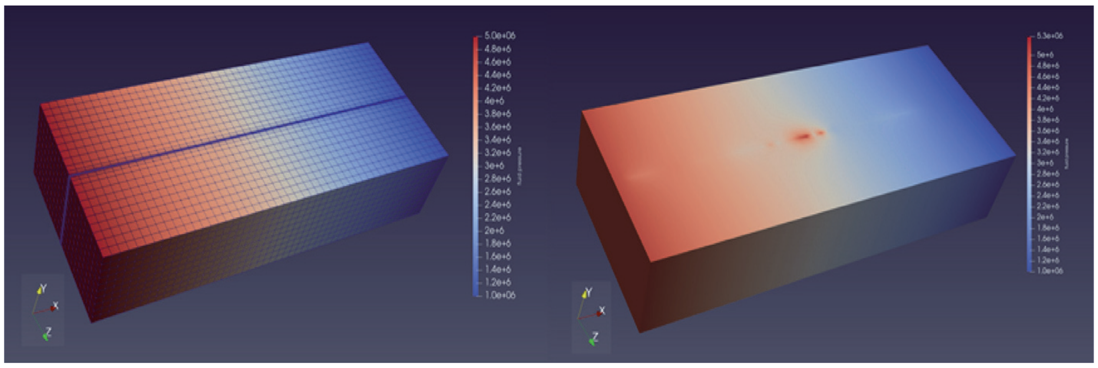

Figure 7.
Left) Result of correct implementation of Dirchlet BC by using Eq. (24). Right) nonphysical (wrong) result due to using the modified linear system (the second approach to implement Dirchlet BC).

On the other hand reduction in size of the linear system by using (24), has great advantage in applications that large number of nodes have Dirichlet (more precisely essential) boundary condition. For example, in pore scale modeling and to approximate the permeability of a sandstone model, we had to set Dirichlet boundary condition for 75% of the nodes. Since the model included almost 10 million elements, solving a linear system with 2.5 million unknowns gave a significant speed up in computational time, in addition to significant improvement in accuracy.

Exercise 10 Robin boundary condition is a combination of Neumann and Dirichlet conditions which states $u - \beta\nu \cdot \kappa\nabla u = g$ on Γ_R. β is a non-zero function (normally a constant number) on Γ_R. Considering $-\nu \cdot \kappa\nabla u = (g-u)/\beta$, explain how to implement it.

5.3 Right hand side term

Function f in (1) is the source or sink term and hence normally is defined on a very small part of the domain. It can be defined as a constant number over an element or can be approximated by its nodal values in the form $f = \sum f_j\varphi_j$ and therefore right hand side term becomes $\sum_{j=1}^{N_n} f_j \int \varphi_j\varphi_i$ with possibly too many $f_j = 0$. We prefer nodal value approximation of f, since in applications that need to solve parabolic equation, f has a value in all elements; see $q^{[k]}$ in Eqs. (27) and (28).

6. Generalization to parabolic equation

In applications such as simulation of compressible flows we need to solve the time dependent equation

$$\frac{\partial\rho(u)}{\partial t} + \nabla \cdot (-\rho(u)\kappa\nabla u) = f(u), \quad \text{in } \Omega \tag{25}$$

where ρ and f are nonlinear functions of u and boundary condition and initial value are provided. A fully implicit time discretization of (25) gives

$$\frac{\rho^{n+1} - \rho^n}{\Delta t} + \nabla \cdot \left(-\rho^{n+1}\kappa\nabla u^{n+1}\right) = f^{n+1}$$

which simplifies to

$$G(u) = \frac{1}{\Delta t}\rho + \nabla \cdot (-\rho\kappa\nabla u) - f(u) - \frac{1}{\Delta t}\rho^n = 0. \tag{26}$$

In Eq. (26), the unknown u-dependent variables ρ and f should be evaluated at the current time t^{n+1}, while ρ^n is known from the previous time step.

Newton–Raphson linearization is the most common method to solve the nonlinear Eq. (26) which generates a sequence $\{u^{[k]}\}$, $k = 0, 1, 2, \ldots$ by solving the following equation

$$\nabla \cdot \left(-\rho^{[k]}\kappa\nabla u\right) + \mu^{[k]}u = q^{[k]}, \tag{27}$$

where

$$\mu^{[k]} = \frac{1}{\Delta t}\frac{d\rho}{du}\bigg|_{u^{[k]}} - \frac{df}{du}\bigg|_{u^{[k]}}, \quad q^{[k]} = \mu^{[k]}u^{[k]} + f^{[k]} - \frac{1}{\Delta t}\rho^n. \tag{28}$$

Having bounded derivatives and a good starting point $u^{[0]}$ (which is normally chosen u^n), the resulted sequence $u^{[k]}$ in Eq. (27) converges quadratically to u^{n+1}, the solution of Eq. (26). Note that (26) is a diffusion–reaction equation which is completely explained how to solve and (core) codes were provided, as well.

7. Conclusion and future works

In this introductory chapter we presented the framework of FEM in brief (but effective) and implemented a numerical solver for diffusion–advection-reaction equation. Accumulation of different terms and setting boundary conditions correctly as well as evaluating of definite integrals without numerical integration were explained and their codes were also given. Finally we showed that nonlinear parabolic equations can be solved by combining of Newton method and diffusion–reaction equation where the latter was explained in this chapter.

However, to have a reliable and efficient simulator several topics and challenges should be addressed. Assuming correct mathematical model, mesh generation and assigning (measured or suggested) values to the coefficients of the problem are very important to make close the numerical model and its solution to the real problem [4, 5]. In a real problem, 2D and 3D elements appear together. For example 2D elements are employed to model faults in a 3D reservoir. Moreover, the generated mesh is normally unstructured and different types of elements are included in the mesh. Hence to assembly the linear system, traversing elements and nodes should have been implemented very efficiently.

Solving linear systems is normally the most time consuming part of the simulation and efficient implementation of linear solvers particularly in parallel machines, are of great interest [6]. Multigrid and multiscale methods, particularly their algebraic form, have attracted interest to solve large scale linear systems since they have shown good scalability in addition to resolving low frequency parts of the error in solving linear systems [7, 8]. Standard Galerkin method that we used in this chapter is not a conservative scheme hence modifications or other methods such as discontinues Galerkin method would be necessary to solve problems in computational fluid dynamics [9, 10]. Proposing and implementing of advanced numerical algorithms to linearize and solve a system of nonlinear coupled initial-boundary value problem in a large scale domain become necessary. At last comparison with real data (if available) and quantifying uncertainties might force us to revisit all steps of the simulation again.

Acknowledgements

Hani Akbari would like to thank Prof. Lars K. Nielsen, scientific director at Novo Nordisk Foundation Center for Bio-Sustainability. Hani Akbari is grateful to DTU-Biosustain since this work was completed when he was a postdoc at DTU-Biosustain.

Author details

Hani Akbari
Shamim Institute of HPC, Scientific Visualization and Machine Learning, Mashhad, Iran

*Address all correspondence to: hani.akbari@shamimhpc.ir

References

[1] Trangenstein J. Numerical Solution of Elliptic and Parabolic Partial Differential Equations. Cambridge University Press; 2013

[2] M. Larson, F. Bengzon, The Finite Element Method: Theory, Implementation, and Applications, Springer, 2013

[3] Ern A, Guermond J. Theory and Practice of Finite Elements. Springer; 2004

[4] Frey P, George PL. Mesh Generation: Application to Finite Elements. second ed. John Wiley & Sons; 2013

[5] Edelsbrunner H. Geometry and Topology for Mesh Generation. Cambridge University Press; 2001

[6] Saad Y. Iterative methods for sparse linear systems. Second ed. SIAM; 2000

[7] Notay Y. Aggregation-based algebraic multigrid for convection-diffusion equations. SIAM J. Sci. Comput. 2012;**34**:A2288-A2316

[8] Akbari H, Pereira F. An algebraic multiscale solver with local Robin boundary value problems for flows in high-contrast media. Journal of Engineering Mathematics. 2020;**123**: 109-128

[9] Zienkiewicz OC, Taylor RL, Nithiarasu P. The Finite Element Method for Fluid Dynamics. 7th ed. Butterworth-Heinemann; 2013

[10] Hesthaven J, Warburton T. Nodal Discontinuous Galerkin Methods. Analysis, and Applications, Springer: Algorithms; 2008