Procedia Computer Science
Volume 80, 2016, Pages 1301–1312
ICCS 2016. The International Conference on Computational Science

DarcyLite: A Matlab Toolbox for Darcy Flow Computation

Jiangguo Liu[1], Farrah Sadre-Marandi[2], and Zhuoran Wang[3] *

[1] Department of Mathematics, Colorado State University (USA), liu@math.colostate.edu
[2] Mathematical Biosciences Institute, Ohio State University (USA), sadre.1@mbi.osu.edu
[3] Department of Mathematics, Colorado State University (USA), wangz@math.colostate.edu

Abstract
DarcyLite is a Matlab toolbox for numerical simulations of flow and transport in 2-dim porous media. This paper focuses on the finite element (FE) methods and the corresponding code modules in DarcyLite for solving the Darcy equation. Specifically, four major types of finite element solvers, the continuous Galerkin (CG), the discontinuous Galerkin (DG), the weak Galerkin (WG), and the mixed finite element methods (MFEM), are examined. Furthermore, overall design and implementation strategies in DarcyLite are discussed. Numerical results are included to demonstrate the usage and performance of this toolbox.

Keywords: Darcy flow, Flow in porous media, Mixed finite element methods, Weak Galerkin

1 Introduction

The Darcy's law is a fundamental equation for modeling flow in porous media. It is usually further coupled with transport equations. Two examples amongst the vast applications in this regard are oil recovery in petroleum reservoirs [4, 10, 13] and drug delivery to tumors. Efficient and robust Darcy solvers are needed for transport simulators [6]. There exist many finite element solvers for the Darcy equation, or more generally second order elliptic boundary value problems, e.g., the continuous Galerkin finite element methods [5], the discontinuous Galerkin finite element methods [2], the enhanced Galerkin (EG) finite element methods [13], the mixed finite element methods [7], and the newly developed weak Galerkin finite element methods [10, 16]. These types of finite element methods have distinctive features in addition to certain common features. Different finite element solvers might be favored by different users and/or applications. Their implementations share some common strategies but also need different treatments [11]. Based on these considerations, we have developed DarcyLite, a Matlab toolbox that contains solvers for the Darcy equation and concentration transport equations. Matlab is chosen as the programming language, due to its popularity, easiness for coding, and integrated

*All three authors were partially supported by US National Science Foundation Grant DMS-1419077.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.485

graphics functionalities. This toolbox is easy to use and efficiently solves medium-size 2-dim flow and transport problems. It is expandable. This allows incorporation of other (new) types of solvers for flow and transport problems. But this paper focuses on the Darcy solvers.

In this paper, we discuss mainly four finite element solvers (CG, DG, WG, MFEM) and their Matlab implementations in `DarcyLite`. This includes details on mesh data structure, quadratures, finite element mass and stiffness matrices, handling of boundary conditions, assembly of global discrete linear systems, computation of numerical velocity and normal fluxes on edges, examination of local mass conservation, and presentation of numerical pressure and velocity. A numerical example is included to demonstrate the use and efficiency of this toolbox.

We consider 2-dim elliptic boundary value problems (Darcy problems) formulated as

$$\begin{cases} \nabla\cdot(-\mathbf{K}\nabla p) \equiv \nabla\cdot\mathbf{u} = f, \quad \mathbf{x}\in\Omega, \\ p = p_D, \ \ \mathbf{x}\in\Gamma^D, \qquad \mathbf{u}\cdot\mathbf{n} = u_N, \ \ \mathbf{x}\in\Gamma^N, \end{cases} \tag{1}$$

where $\Omega\subset\mathbb{R}^2$ is a bounded polygonal domain, p is the primal unknown (pressure), $\mathbf{K}$ is a hydraulic conductivity (permeability) tensor that is uniformly symmetric positive-definite, f is a source term, p_D, u_N are respectively Dirichlet and Neumann boundary data, $\mathbf{n}$ the unit outward normal vector on $\partial\Omega$, which has a nonoverlapping decomposition $\Gamma^D\cup\Gamma^N$.

We define a subspace and a manifold respectively for scalar-valued functions as follows

$$H_{D,0}(\Omega) = \{p\in H^1(\Omega) : p|_{\Gamma^D} = 0\}, \qquad H_{D,p_D}(\Omega) = \{p\in H^1(\Omega) : p|_{\Gamma^D} = p_D\}.$$

The variational form for the primal variable pressure reads as: Seek $p\in H^1_{D,p_D}(\Omega)$ such that

$$\int_\Omega \mathbf{K}\nabla p\cdot\nabla q = \int_\Omega fq - \int_{\Gamma_N} u_N q, \qquad \forall q\in H^1_{D,0}(\Omega). \tag{2}$$

The Darcy equation can also be rewritten as a system of two first-order equations by considering the primal variable (pressure) and flux (velocity $\mathbf{u} = -\mathbf{K}\nabla p$) as follows

$$\mathbf{K}^{-1}\mathbf{u} + \nabla p = \mathbf{0}, \qquad \nabla\cdot\mathbf{u} = f. \tag{3}$$

We define a subspace and a manifold respectively for vector-valued functions as follows

$$\begin{aligned} &H_{N,0}(\mathrm{div},\Omega) = \{\mathbf{v}\in L^2(\Omega)^2 : \mathrm{div}\mathbf{v}\in L^2(\Omega), \mathbf{v}|_{\Gamma^N} = \mathbf{0}\}, \\ &H_{N,u_N}(\mathrm{div},\Omega) = \{\mathbf{v}\in L^2(\Omega)^2 : \mathrm{div}\mathbf{v}\in L^2(\Omega), \mathbf{v}|_{\Gamma^N}\cdot\mathbf{n} = u_N\}. \end{aligned}$$

The mixed variational formulation reads as: Seek $\mathbf{u}\in H_{N,u_N}(\mathrm{div},\Omega)$ and $p\in L^2(\Omega)$ such that

$$\begin{cases} \displaystyle\int_\Omega (\mathbf{K}^{-1}\mathbf{u})\cdot\mathbf{v} - \int_\Omega p(\nabla\cdot\mathbf{v}) = -\int_{\Gamma_D} p_D\mathbf{v}\cdot\mathbf{n}, & \forall\mathbf{v}\in H_{N,0}(\mathrm{div},\Omega), \\ \displaystyle -\int_\Omega (\nabla\cdot\mathbf{u})q = -\int_\Omega fq, & \forall q\in L^2(\Omega). \end{cases} \tag{4}$$

For Darcy FE solvers, two desired properties are (assuming $\mathbf{u}_h$ is the numerical velocity):

- **Local mass conservation**: For any element E with $\mathbf{n}$ being the outward unit normal vector on its boundary ∂E, there holds

$$\int_{\partial E} \mathbf{u}_h\cdot\mathbf{n} = \int_E f. \tag{5}$$

- **Normal flux continuity (in the integral form)**: For an interior edge γ shared by two elements E_i that have respectively outward unit normal vectors $\mathbf{n}_i (i = 1, 2)$, there holds

$$\int_\gamma \mathbf{u}_h|_{E_1} \cdot \mathbf{n}_1 + \int_\gamma \mathbf{u}_h|_{E_2} \cdot \mathbf{n}_2 = 0. \tag{6}$$

These important quantities are set as standard outputs of the FE solvers in this toolbox:

(1) *Numerical pressure average* over each element;

(2) *Numerical velocity* at each element center (and if needed, coefficients in the basis of the elementwise approximation subspace, e.g., RT_0 or $RT_{[0]}$);

(3) *Outward normal fluxes* on all edges of each element;

(4) *Local-mass-conservation residual* on each element if a FE solver is not locally conservative;

(5) *Normal flux discrepancy* across each interior edge if the normal fluxes are not continuous.

2 FE Solvers for Darcy Equation: CG, DG, WG, MFEM

2.1 Continuous Galerkin (CG)

The CG solvers for the Darcy equation are based on discretizations of the primal variational formulation (2). CG P_1 on triangular meshes and CG Q_1 on rectangular meshes are most frequently used. For a triangular mesh $\mathcal{T}_h$ with no hanging nodes, let V_h be the space of all continuous piecewise linear polynomials on $\mathcal{T}_h$. Let $V_0^h = V_h \cap H_{D,0}(\Omega)$ and $V_D^h = V_h \cap H_{D,p_D}(\Omega)$ (after Lagrangian interpolation of Dirichlet boundary data being performed). A finite element scheme using CG P_1 for the pressure is established as: Seek $p_h \in V_D^h$ such that

$$\sum_{T \in \mathcal{T}_h} \int_E \mathbf{K}\nabla p_h \cdot \nabla q = \sum_{T \in \mathcal{T}_h} \int_T fq - \sum_{\gamma \in \Gamma_h^N} \int_\gamma u_N q, \qquad \forall q \in V_0^h. \tag{7}$$

After the numerical pressure p_h is obtained, the numerical velocity is calculated *ad hoc* [4].

The CG P_1 basis functions are actually barycentric coordinates. Computation and assembly of element stiffness matrices are relatively easy. CG P_1 or Q_1 schemes have the least numbers of unknowns. But CG schemes are not locally mass-conservative and do not have continuous normal fluxes across edges [8]. Post-processing is needed [5] to render a numerical velocity the above two properties.

2.2 Discontinuous Galerkin (DG)

Let $\mathcal{E}_h$ be a rectangular or triangular mesh, define V_h^k as the space of (discontinuous) piecewise polynomials with a total degree $\leq k$ on $\mathcal{E}_h$. A DG scheme seeks $p_h \in V_h^k$ such that [13]

$$\mathcal{A}_h(p_h, q) = \mathcal{F}(q), \quad \forall q \in V_h^k, \tag{8}$$

where

$$\begin{aligned}\mathcal{A}_h(p_h, q) = & \sum_{E \in \mathcal{E}_h} \int_E \mathbf{K}\nabla p_h \cdot \nabla q \; - \sum_{\gamma \in \Gamma_h^I \cup \Gamma_h^D} \int_\gamma \{\mathbf{K}\nabla p_h \cdot \mathbf{n}\}[q] \\ & + \beta \sum_{\gamma \in \Gamma_h^I \cup \Gamma_h^D} \int_\gamma \{\mathbf{K}\nabla q \cdot \mathbf{n}\}[p_h] \; + \sum_{\gamma \in \Gamma_h^I \cup \Gamma_h^D} \frac{\alpha_\gamma}{h_\gamma} \int_\gamma [p_h][q],\end{aligned} \tag{9}$$

$$\mathcal{F}(q) = \sum_{E\in\mathcal{E}_h} \int_E fq - \sum_{\gamma\in\Gamma_h^N} \int_\gamma u_N q + \beta \sum_{\gamma\in\Gamma_h^D} \int_\gamma \mathbf{K}\nabla q \cdot \mathbf{n} p_D + \sum_{\gamma\in\Gamma_h^D} \frac{\alpha_\gamma}{h_\gamma} \int_\gamma p_D q. \tag{10}$$

Here $\alpha_\gamma > 0$ is a penalty factor for any edge $\gamma \in \Gamma_h$ and β is a formulation parameter [13]. Depending on the choice of β, one ends up with the symmetric interior penalty Galerkin (SIPG) for $\beta = -1$, the nonsymmetric interior penalty Galerkin (NIPG) for $\beta = 1$, and the incomplete interior penalty Galerkin (IIPG) for $\beta = 0$.

Stability of the above DG scheme for a numerical pressure relies on the penalty factor. DG numerical velocity can be calculated *ad hoc*. The DG velocity is locally mass-conservative, but its normal component is not continuous across element boundaries. As pointed out in [2], this leads to nonphysical oscillations, if the DG velocity is coupled to a DG transport solver in a straightforward manner. This could make particle tracking difficult or impossible, if the DG velocity is used in a characteristic-based transport solver.

The features and usefulness of the $H(\text{div})$ finite elements, especially, the BDM finite element spaces motivate post-processing of the DG velocity via projection [2]. The post-processed velocity has the following properties [2]:

(i) The new numerical velocity has continuous normal components on element interfaces;

(ii) The new numerical velocity reproduces the averaged normal flux of the DG velocity;

(iii) It has the same accuracy and convergence order as the original DG velocity.

Details on implementation of the DG schemes and their post-processing can be found in [2, 11].

2.3 Weak Galerkin (WG)

The WG finite element methods are developed based on the innovative concepts of weak gradients and discrete weak gradients [16]. Instead of using the *ad hoc* gradient of shape functions, discrete weak gradients are established at the element level to provide nice approximations of the classical gradient in partial differential equations.

The WG approach considers a discrete shape function in two pieces: $v = \{v^\circ, v^\partial\}$, the interior part v° could be a polynomial of degree $l \geq 0$ in the interior E° of an element, the boundary part v^∂ on E^∂ could be a polynomial of degree $m \geq 0$, its discrete weak gradient $\nabla_{w,n} v$ is specified in $V(E,n) \subseteq P_n(E)^2 (n \geq 0)$ via integration by parts:

$$\int_E (\nabla_{w,n} v) \cdot \mathbf{w} = \int_{\partial E} v^\partial (\mathbf{w} \cdot \mathbf{n}) - \int_E v^\circ (\nabla \cdot \mathbf{w}), \qquad \forall w \in V(E,n).$$

For example, for a triangle T, $WG(P_0, P_0, RT_0)$ uses a constant basis function on T°, a constant basis function on each edge on ∂T, and specifies the discrete weak gradients of these 4 basis functions in RT_0. Normalized basis functions for RT_0 facilitate computation of these discrete weak gradients [8, 10, 11], although small-size linear systems need to be solved in general.

The WG schemes for the Darcy equation rely on discretizations of (2). Let $\mathcal{E}_h$ be a triangular or rectangular mesh, l, m, n nonnegative integers, $S_h(l,m)$ the space of discrete shape functions on $\mathcal{E}_h$ that have polynomial degree l in element interior and degree m on edges, $S_h^0(l,m)$ the subspace of functions in $S_h(l,m)$ that vanish on Γ^D. Seek $p_h = \{p_h^\circ, p_h^\partial\} \in S_h(l,m)$ such that $p_h^\partial|_{\Gamma^D} = Q_h^\partial p_D$ (projection of Dirichlet boundary data) and

$$\mathcal{A}_h(p_h, q) = \mathcal{F}(q), \qquad \forall q = \{q^\circ, q^\partial\} \in S_h^0(l,m), \tag{11}$$

$$\mathcal{A}_h(p_h, q) := \sum_{E\in\mathcal{E}_h} \int_E \mathbf{K}\nabla_{w,n} p_h \cdot \nabla_{w,n} q, \qquad \mathcal{F}(q) := \sum_{E\in\mathcal{E}_h} \int_E f q^\circ - \sum_{\gamma\in\Gamma_h^N} \int_\gamma u_N q. \tag{12}$$

After obtaining the numerical pressure p_h, one computes the WG numerical velocity by

$$\mathbf{u}_h = R_h(-\mathbf{K}\nabla_{w,n} p_h), \tag{13}$$

where R_h is the local L^2-projection onto $V(E, n)$. But it can be skipped when $\mathbf{K}$ is a constant scalar on the element. The normal fluxes are computed accordingly. The WG schemes are locally conservative and produce continuous normal fluxes (in the integral form) [10, 11, 16].

2.4 Mixed Finite Element Methods (MFEM)

The mixed finite element schemes for the Darcy equation are based on discretizations of the mixed variational form (4). A pair of finite element spaces (V_h, W_h) are chosen such that $V_h \subset H(\text{div}, \Omega)$ for approximating velocity and $W_h \subset L^2(\Omega)$ for approximating pressure and together they satisfy the inc-sup condition. Among the popular choices are $(RT_0, P_0), (BDM_1, P_0)$ for triangular meshes and $(RT_{[0]}, Q_0)$ for rectangular meshes [10, 11].

For a rectangular or triangular mesh $\mathcal{E}_h$, denote $U_h = V_h \cap H_{u_N,N}(\text{div}; \Omega)$ and $V_h^0 = V_h \cap H_{0,N}(\text{div}; \Omega)$. A mixed FE scheme can be stated as: Seek $\mathbf{u}_h \in U_h$ and $p_h \in W_h$ such that

$$\begin{cases} \sum_{E\in\mathcal{E}_h} \int_E \mathbf{K}^{-1}\mathbf{u}_h \cdot \mathbf{v} - \sum_{E\in\mathcal{E}_h} \int_E p_h(\nabla \cdot \mathbf{v}) = -\sum_{\gamma\in\Gamma_h^D} \int_\gamma p_D(\mathbf{v}\cdot\mathbf{n}), & \forall \mathbf{v} \in V_h^0, \\ -\sum_{E\in\mathcal{E}_h} \int_E (\nabla\cdot\mathbf{u}_h) q = -\sum_{E\in\mathcal{E}_h} \int_E f q, & \forall q \in W_h. \end{cases} \tag{14}$$

For implementation of (RT_0, P_0), the edge-based basis functions for V_h can be used in computation and assembly of element matrices [10, 11]. A symmetric indefinite linear system is solved. This is taken care by Matlab backslash, although it is nontrivial behind the scene.

An obvious advantage of the MFEM schemes is the automatic local mass conservation (obtained by taking test function $q = 1$ in (14) 2nd equation) and normal flux continuity (built in the construction of $H(\text{div})$ finite element spaces). It is interesting to observe certain equivalence between the MFEM schemes and the WG finite element schemes [10].

3 Design and Implementation of `DarcyLite`

`DarcyLite` is implemented in Matlab, which is a popular interpretative language. Although it is different than compilation languages like C/C++ or Fortran, Matlab is very efficient in matrix or array operations, since its core was implemented mainly in Fortran. This also means that Matlab arrays are stored columnwise. The integrated development environment and graphics functionalities offered by Matlab are very helpful for educational purposes.

The above observations guide our overall design of `DarcyLite`:

- *Code modules and separation of tasks*;
- *Minimization of function calls*;
- *Columnwise storage and access for full matrices or arrays*;

- *Utilizing Matlab (i, j, s)-format for sparse matrices*;
- *Performing operations on a mesh for all elements simultaneously.*

This section further elaborates on these ideas through a list of selected topics. Some data structures and implementation techniques in `DarcyLite` share the same spirit as those in [1, 3].

3.1 Mesh Data Structure

Mesh information can be categorized as geometric (node coordinates, positions of element centers, etc.), topological (an element vs its nodes, adjacent edges of a given edge, etc.), and combinatorial (the number of neighboring nodes of a given node, etc.). Some basic mesh data are used by all types of finite element solvers, even though they use some derived mesh info in different ways. Mesh data can be organized as

(i) *Primary* info such as the numbers of nodes/elements, node coordinates, an element vs its nodes: `NumNds, NumEms, node, elem`

(ii) *Secondary* info such as an edge vs its vertices, an edge vs its neighboring element(s), an element vs its all edges: `NumEgs, edge, edge2elem, elem2edge, LenEg, area, EmCntr`

(iii) *Tertiary* info are more specific to particular finite element solvers. For instance, DG needs to know what are the neighboring elements for a give element, MFEM and WG need specific info on whether an edge is the 2nd edge of a given element. The info on the (unit) normal vector of a given edge is also used by MFEM and WG.

For example, the geometric info about mesh node coordinates is organized as an array of size `NumNds*2`, the topological info about triangular elements vs their vertices is organized as an array of size `NumEms*3`. Shown below is a code fragment for computing all triangle areas based on the columnwise storage in Matlab.

```
k1 = TriMesh.elem(:,1);  k2 = TriMesh.elem(:,2);  k3 = TriMesh.elem(:,3);
x1 = TriMesh.node(k1,1);  x2 = TriMesh.node(k2,1);  x3 = TriMesh.node(k3,1);
y1 = TriMesh.node(k1,2);  y2 = TriMesh.node(k2,2);  y3 = TriMesh.node(k3,2);
TriMesh.area = 0.5*abs((x2-x1).*(y3-y1)-(x3-x1).*(y2-y1));
```

Based on these three levels of mesh info, `DarcyLite` organizes mesh data as a structure that has several dynamic fields. The fields can be easily added or removed. Code modules are provided for mesh info enrichment (adding data fields at a higher level), see code modules

```
TriMesh_Enrich1.m,  TriMesh_Enrich3.m,  RectMesh_Enrich1.m,  RectMesh_Enrich3.m
```

3.2 Implementation of Quadratures

Integrals on elements and element interfaces comprise a major portion of computation in finite element methods. Although this looks like a simple issue, but code efficiency can be improved when we adopt an unconventional approach. For example, in the $WG(P_0, P_0, RT_0)$ scheme for the Darcy equation, one needs to compute the integrals $\int_T f$ over all triangular elements. If a K-point Gaussian quadrature is applied, then the conventional approach would be

$$\int_T f dT \approx |T| \sum_{k=1}^{K} f(\alpha_k P_1 + \beta_k P_2 + \gamma_k P_3) w_k, \quad \forall T \in \mathcal{T}_h,$$

where $(\alpha_k, \beta_k, \gamma_k), w_k (k = 1, \dots, N)$ are respectively the barycentric coordinates and weights of the quadrature points. The order of summation and loop can be reversed:

```
GlbRHS = zeros(DOFs,1);
NumQuadPts = size(GAUSSQUAD.TRIG,1);
for k=1:NumQuadPts
  qp = GAUSSQUAD.TRIG(k,1) * TriMesh.node(TriMesh.elem(:,1),:)...
     + GAUSSQUAD.TRIG(k,2) * TriMesh.node(TriMesh.elem(:,2),:)...
     + GAUSSQUAD.TRIG(k,3) * TriMesh.node(TriMesh.elem(:,3),:);
  GlbRHS(1:NumEms) = GlbRHS(1:NumEms) + GAUSSQUAD.TRIG(k,4) * EqnBC.fxnf(qp);
end
GlbRHS(1:NumEms) = GlbRHS(1:NumEms) .* TriMesh.area;
```

3.3 Element-level Small Matrices and Mesh-level Sparse Matrices

Finite element schemes involve computation of element-level small-size full matrices, e.g., mass matrices and stiffness matrices, and mesh-level large-size sparse matrices, e.g., the global stiffness matrix. In general, we avoid using cell arrays of usual matrices, since they may be inefficient. Instead we use 3-dim arrays. For example, in the $WG(P_0, P_0, RT_0)$ scheme for the Darcy equation, each triangle involves one WG basis function for element interior and three WG basis functions for the edges. The discrete weak gradient of each of the four WG basis functions is a linear combination of the three RT_0 normalized basis functions [10]. We organize these coefficients as a three-dimensional array

```
CDWGB = zeros(TriMesh.NumEms, 4, 3);
```

The interaction of the discrete weak gradients of the 3 basis functions results in a 3-dim array

```
ArrayGG = zeros(TriMesh.NumEms, 3, 3);
```

These element-level small full matrices are assembled into the sparse global stiffness matrix

```
DOFs = TriMesh.NumEms + TriMesh.NumEgs;
GlbMat = sparse(DOFs,DOFs);
```

Instead of using a loop over all elements, we utilize the (i, j, s)-structure in Matlab and the mesh topological info:

```
for i=1:3
  II = TriMesh.NumEms + TriMesh.elem2edge(:,i);
  for j=i:3  % Utilizing symmetry
    JJ = TriMesh.NumEms + TriMesh.elem2edge(:,j);
    if (j==i)
      GlbMat = GlbMat + sparse(II,II,ArrayGG(:,i,i),DOFs,DOFs);
    else
      GlbMat = GlbMat + sparse(II,JJ,ArrayGG(:,i,j),DOFs,DOFs);
      GlbMat = GlbMat + sparse(JJ,II,ArrayGG(:,i,j),DOFs,DOFs);
    end
  end
end
```

See the following code modules for more details:

```
Darcy_WG_TriP0P0RT0_AsmSlv.m,  Darcy_WG_RectQ0P0RT0_AsmSlv.m
```

3.4 Enforcement of Essential Boundary Conditions

For CG and WG, Dirichlet boundary conditions are essential, but Neumann boundary conditions are natural, see Equations (7,11). For MFEM, Neumann conditions are essential but Dirichlet conditions are natural, see Equation (14). For DG, Dirichlet conditions are enforced weakly [13] in the bilinear and linear forms, see Equations (8,9,10). Theoretically, a zero essential boundary condition corresponds to a nice subspace for the numerical solution in the finite element scheme. For a non-zero essential condition, however, we usually use the terminology "manifold" to accommodate the numerical solution. From the implementation viewpoint, to enforce essential boundary conditions, we need to modify the global sparse linear system obtained from a finite element scheme. There are basically two approaches, hard ("brutal") and soft ("gentle"). We illustrate the latter using a simple linear system.

Let $\{x_1, x_2\}$ be unknowns that satisfy a linear system as follows

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Assume $x_2 = c$ is the essential boundary condition. A brutal way is to modify the system to

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ c \end{bmatrix}.$$

But this usually damages the symmetry in the original linear system. An alternative is to consider solving $A_{11}x_1 = b_1 - A_{12}c$, which is a linear system with a smaller size and maintains the symmetry of A_{11} shown in the original system. Note that

$$\begin{bmatrix} b_1 - A_{12}c \\ * \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} 0 \\ c \end{bmatrix}$$

involves a simple matrix-vector multiplication (using the original global coefficient matrix) and we just need to take the first block in the final result vector. See the following code modules for more details:

```
Darcy_CG_RectQ1t_AsmSlv.m, Darcy_WG_TriP0P0RT0_AsmSlv.m, Darcy_WG_RectQ0P0RT0_AsmSlv.m
```

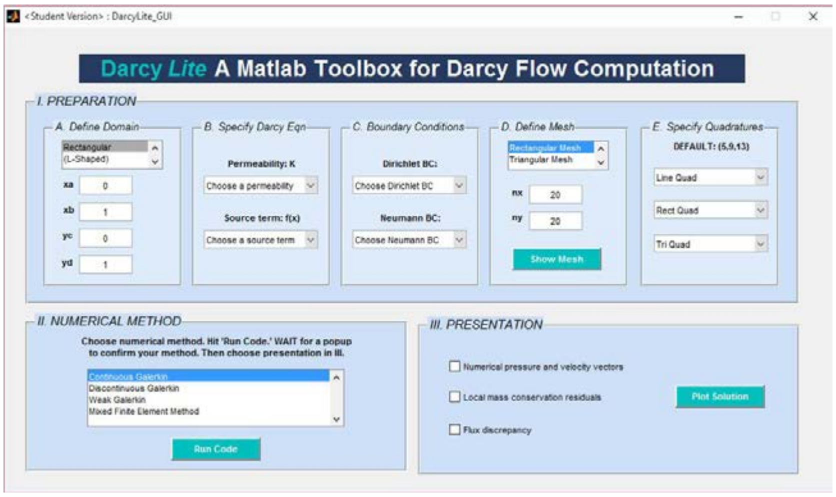
3.5 Interface with Other Software Packages

`DarcyLite` focuses on solving flow and transport problems in the environment provided by Matlab. It can import triangular meshes generated by other packages, e.g., `PDE Toolbox` and `DistMesh` [12], which also run on Matlab. `DarcyLite` provides functions for triangular mesh conversion from the aforementioned two packages.

`Triangle` is a popular triangular mesh generator developed in C. `DarcyLite` can read in the mesh data files generated by `Triangle` in the so-called flat file transfer (FFT) approach.

3.6 Graphical User Interface (GUI) of `DarcyLite`

A graphical user interface (GUI) is provided with `DarcyLite` for demonstration and education purposes. It was developed using Matlab built-in `guide` (graphical user interface development environment). The GUI is based on the event-driven design principle and some techniques in SENSAI [14] are adopted. The GUI has three parts as shown in Figure 1: Preparation, Numerical method, and Presentation.



xa 0
xb 1
yc 0
yd 1

nx 20
ny 20

Choose numerical method. Hit 'Run Code.' WAIT for a popup to confirm your method. Then choose presentation in III.

Figure 1: A screen snapshot of the Graphical User Interface (GUI) for `DarcyLite`

I. Preparation. This part allows the user to specify details for a problem to be solved.

(A) A domain is defined, including the endpoints along the x- and y-axis. The default settings suggest a rectangular domain with both axes starting at 0 and ending at 1.

(B) Specify the Darcy equation thru a hydraulic conductivity K and a source term f.

(C) Dirichlet and Neumann boundary conditions are specified. The two built-in options for K, f, and boundary conditions are defined according to two popular test examples.

(D) A mesh is to be generated. The user can choose between a rectangular or triangular mesh by specifying the numbers of uniform partitions nx, ny for the x, y-directions, respectively. Default settings suggest $nx = 20, ny = 20$. The $\boxed{\text{Show Mesh}}$ button pops up a figure window for the user to check whether a correct mesh has been generated.

(E) Gaussian type quadratures (for edges, rectangles, and triangles) are chosen. The default choices (5,25,13) are sufficient for most cases.

II. Numerical Method. This part of the GUI lists the four major types of finite element solvers. The user can choose among Continuous Galerkin, Discontinuous Galerkin, Weak Galerkin, or Mixed Finite Element Method. Then the user must click the $\boxed{\text{Run Code}}$ button. A popup will confirm that the inputs from Part I are correct for the chosen finite element solver. Otherwise, an error message will pop up.

III. Presentation. The checkboxes in this part allow the user to display any combination of the numerical pressure and velocity profiles, local mass-conservation residual (LMCR), and flux discrepancy across edges.

4 A Transport Solver: Implicit Euler + Weak Galerkin

`DarcyLite` contains also finite element solvers for transport problems in 2-dim prototyped as

$$\begin{cases} c_t + \nabla\cdot(\mathbf{v}c - D\nabla c) = f(x,y,t), \quad (x,y)\in\Omega, t\in(0,T), \\ c(x,y,t) = 0, \quad (x,y)\in\partial\Omega, t\in(0,T), \\ c(x,y,0) = c_0(x,y), \quad (x,y)\in\Omega. \end{cases} \tag{15}$$

Here $c(x,y,t)$ is the unknown (solute) concentration, $\mathbf{v}$ the Darcy velocity, $D>0$ a diffusion constant, f a source/sink term. There exist various types of finite element methods for transport problems. Here we briefly discuss a numerical scheme that utilizes the implicit Euler for time-marching and weak Galerkin for spatial approximation.

For simplicity, we assume Ω is a rectangular domain equipped with a rectangular mesh $\mathcal{E}_h$ and a numerical velocity $\mathbf{u}_h$ has been obtained from applying a finite element solver that is locally mass-conservative and has continuous normal fluxes. The unknown concentration is approximated using the lowest order finite element space $WG(Q_0,P_0,RT_{[0]})$. Let $C_h^{(n)} (n\ge 1)$ be such an approximation at discrete time t_n, then there holds for $n\ge 1$,

$$\begin{aligned} &\sum_{E\in\mathcal{E}_h}(C_h^{(n)},w)_E - \Delta t\sum_{E\in\mathcal{E}_h}(\mathbf{u}_h C_h^{(n)},\nabla_{w,d}w)_E + \Delta t\; D\sum_{E\in\mathcal{E}_h}(\nabla_{w,d}C_h^{(n)},\nabla_{w,d}w)_E \\ &\quad = \sum_{E\in\mathcal{E}_h}(C_h^{(n-1)},w)_E + \Delta t\sum_{E\in\mathcal{E}_h}(f,w)_E. \end{aligned} \tag{16}$$

An initial approximant $C_h^{(0)}$ can be obtained via local L_2-projection of $c_0(x,y)$ into the WG finite element space.

The "Implicit Euler + Weak Galerkin" scheme has two nice properties:

(i) $C_h^{(n)}|_{E^\circ}$ represents intuitively the *cell average of concentration* on any element E.

(ii) The scheme is *locally* and hence *globally conservative*. This is verified by taking a test function w that has value 1 in one element interior but 0 in all others and on all edges.

The 2nd term on the left side of the above equation characterizes interaction of the flow (Darcy velocity) and the concentration (discrete weak) gradient. The 3rd term is a symmetric term similar to that in any elliptic problem. The 1st term on the right side represents the mass at the previous time moment, whereas the last term depicts the source/sink contribution during the time period $[t_{n-1},t_n]$.

5 Numerical Experiments

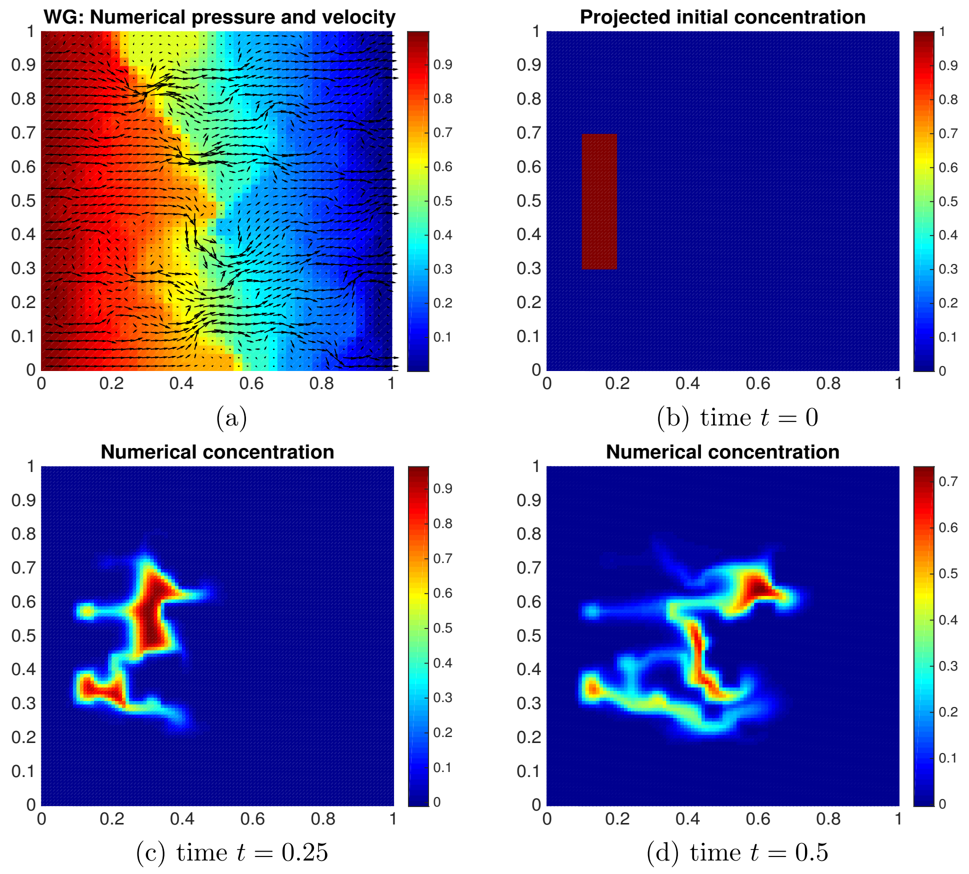
This section presents numerical results to demonstrate the use of `DarcyLite`. We consider an example of coupled flow and transport. A numerical Darcy velocity is fed into the transport solver discussed in the previous section.

For the Darcy equation, $\Omega=(0,1)^2$, the hydraulic conductivity profile is taken from [7] (also used in [10, 11]). There is no source. A Dirichlet condition $p=1$ is specified for the left boundary and $p=0$ for the right boundary. A zero Neumann condition is set for the lower and upper boundaries. The Darcy solver $WG(Q_0,P_0,RT_{[0]})$ is used on a uniform 100×100 rectangular mesh (but for graphics clarity, Figure 2 Panel (a) shows results of 40×40 mesh).

For the transport problem (15), $T = 0.5$, $D = 10^{-3}$, there is no source. An initial constant concentration is placed in $[0.1, 0.2] \times [0.3, 0.7]$. The $RT_{[0]}$ numerical velocity from the Darcy solver is used in the IE+WG scheme (16) with a 100×100 rectangular mesh and $\Delta t = 5 * 10^{-4}$.

Figure 2 Panel (b)(c)(d) present numerical concentration profiles for time moments $t = 0, 0.25, 0.5$, respectively. It can be observed that the transport pattern reflects well the flow features: (i) The upper part of the domain has stronger flow and hence the concentration front in the upper part advances faster than that in the lower part. (ii) In Panel (d), in the middle of the domain, an almost vertical transport path can be observed. This clearly reflects the strong flow in the domain from position $(0.4, 0.5)$ downward to position $(0.5, 0.3)$, see Panel (a) also.

Note that the mass flux on the domain boundary is zero. It can be checked numerically that the total mass in the domain $\sum_{E\in\mathcal{E}_h} C_h^{(n)}|_{E^\circ}|E|$ remains at $4.000 * 10^{-2}$ for all discrete time moments ($t = 0$ through $t = 0.5$ with increment 0.05). This verifies the mass conservation property (ii) discussed in Section 4. However, there are small negative concentrations. Eliminating oscillations in numerical concentrations of convection-dominated transport problems is a nontrivial issue [9]. Particular treatments for the "Implicit Euler + Weak Galerkin" transport solver are currently under our investigation.



(a) (b) time $t = 0$ (c) time $t = 0.25$ (d) time $t = 0.5$

Figure 2: Example: Coupled Darcy flow and transport. (a) Numerical pressure and velocity. (b) Initial concentration. (c) Concentration at time $t = 0.25$. (d) Concentration at time $t = 0.5$. Results for (c)(d) are obtained using $WG(Q_0, P_0, RT_{[0]})(h = 10^{-2})$ and implicit Euler $\Delta t = 5 * 10^{-4}$.

6 Concluding Remarks

`DarcyLite` is a small-size code package developed in Matlab for solving 2-dim flow and transport equations. It will be extended to include 3-dim solvers and test cases.

For the flow equation, `DarcyLite` provides four major types of FE solvers (CG, DG, WG, MFEM) on triangular and rectangular meshes. CG post-processing [5] and the enhanced Galerkin (EG) [13] will be included later. For the transport equation, `DarcyLite` provides solvers for both steady-state and transient problems. For the latter, it offers solvers of Eulerian type and Eulerian-Lagrangian type [15]. For coupled flow and transport problems, solvers for a 2-phase model [8] are provided. `DarcyLite` is being extended to include more solvers that are efficient, robust, and respect physical properties, e.g., local conservation, positivity-preserving.

Besides applications like petroleum reservoir and groundwater simulations, `DarcyLite` is being extended to include flow and transport problems in biological media, e.g., drug delivery.

The URL for this code package is `http://www.math.colostate.edu/~liu/code.html`

References

[1] J. Alberty, C. Carstensen, and S. Funken. Remarks around 50 lines of matlab: short finite element implementation. *Numer. Algor.*, 20:117–137, 1999.

[2] P. Bastian and B. Riviere. Superconvergence and $h(div)$ projection for discontinuous galerkin methods. *Int. J. Numer. Meth. Fluids*, 42:1043–1057, 2003.

[3] L. Chen. ifem: an integrated finite element methods package in matlab. *Tech. Report, Math Dept., Univ. of California at Irvine (2009)*, 2009.

[4] Z. Chen, G. Huan, and Y. Ma. *Computational methods for multiphase flows in porous media.* SIAM, 2006.

[5] B. Cockburn, J. Gopalakrishnan, and H. Wang. Locally conservative fluxes for the continuous galerkin method. *SIAM J. Numer. Anal.*, 45:1742–1770, 2007.

[6] C. Dawson, S. Sun, and M. Wheeler. Compatible algorithms for coupled flow and transport. *Comput. Meth. Appl. Mech. Engrg.*, 193:2565–2580, 2004.

[7] L. Durlofsky. Accuracy of mixed and control volume finite element approximations to darcy velocity and related quantities. *Water Resour. Res.*, 30:965–973, 1994.

[8] V. Ginting, G. Lin, and J. Liu. On application of the weak galerkin finite element method to a two-phase model for subsurface flow. *J. Sci. Comput.*, 66:225–239, 2016.

[9] V. John and E. Schmeyer. Finite element methods for time-dependent convection–diffusion–reaction equations with small diffusion. *Comput. Meth. Appl. Mech. Engrg.*, 198:475–494, 2008.

[10] G. Lin, J. Liu, L. Mu, and X. Ye. Weak galerkin finite element methdos for darcy flow: Anistropy and heterogeneity. *J. Comput. Phys.*, 276:422–437, 2014.

[11] G. Lin, J. Liu, and F. Sadre-Marandi. A comparative study on the weak galerkin, discontinuous galerkin, and mixed finite element methods. *J. Comput. Appl. Math.*, 273:346–362, 2015.

[12] P. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46:329–345, 2004.

[13] S. Sun and J. Liu. A locally conservative finite element method based on piecewise constant enrichment of the continuous galerkin method. *SIAM J. Sci. Comput.*, 31:2528–2548, 2009.

[14] S. Tavener and M. Mikucki. Sensai: A matlab package for sensitivity analysis. http://www.math.colostate.edu/ tavener/FEScUE/SENSAI/sensai.shtml.

[15] H. Wang, H.K. Dale, R.E. Ewing, M.S. Espedal, R.C. Sharpley, and S. Man. An ellam scheme for advection-diffusion equations in two dimensions. *SIAM J. Sci. Comput.*, 20:2160–2194, 1999.

[16] J. Wang and X. Ye. A weak galerkin finite element method for second order elliptic problems. *J. Comput. Appl. Math.*, 241:103–115, 2013.