Journal of Computational Physics 513 (2024) 113169

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A flux-based moving mesh method applied to solving the Poisson–Nernst–Planck equations

Minrui Lv [a,b], Benzhuo Lu [a,b,*]

[a] LSEC, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
[b] School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China

ARTICLE INFO

Keywords:
Moving mesh method
Poisson–Nernst–Planck equations
Finite element method

ABSTRACT

The moving mesh method is one of the important adaptive mesh methods which is practically useful when the mesh size and overall resolution are provided and fixed as seen in many engineering computations. We employ a moving mesh technique combined with a finite element method (FEM) to solve a widely-used charged carrier transport model, the Poisson–Nernst–Planck (PNP) equations, the solutions of which often have boundary or internal layers and sharp interfaces when the convection is dominated. Considering that flux is a significant physical quantity in designing stable and accurate numerical algorithm for transport problems such as the PNP system, we start from a relatively general functional and propose a flux-based monitor function and an adaptive step size-controlling algorithm for guiding the mesh movement in the FEM solution of the PNP equations. The numerical results illustrate that the moving mesh method is effectively addressing challenges arising from solution singularities and the convection-dominated effects. It also shows that the moving mesh finite element method we propose exhibits superior performance compared to both the traditional moving mesh finite element method and fixed mesh finite element method in some scenarios. Furthermore, it demonstrates better adherence to the physical properties of energy dissipation inherent in the PNP equation.

1. Introduction

The PNP model is a well-known carrier transport model that takes into account the movement of charged particles influenced by both the Brownian motion of free particles and the total electric field. The total electric field includes external and self-generated electric fields varying with particle motion. This model is useful in simulating biological ion channels [31,38,45,55], semiconductor devices [7,49], and electrochemical systems [14,52]. To solve the PNP equations numerically, there are various methods available, including finite element [27,57,61,62,66,67], finite difference [22,26,29,41,42], and finite volume methods [3,11,48]. The finite element method is particularly suitable to handle complex shapes and boundary conditions.

Solving the PNP equations is a challenging task due to its inherent nonlinearity. The Nernst-Planck equation is a nonlinear convection-diffusion equation where the convection velocity vector is influenced by the electric potential gradient. This causes the formation of boundary or interior layers in regions with a higher rate of change in electric potential than the diffusion coefficient. Therefore, considerable efforts have been devoted to dealing with the challenge. Various techniques have been explored, including stabilization techniques [2,9], exponential fitting method [4,8], inverse-average-type finite element [47,60,65,67]. Aside from those

* Corresponding author.
E-mail address: bzlu@lsec.cc.ac.cn (B. Lu).

https://doi.org/10.1016/j.jcp.2024.113169
Received 1 September 2023; Received in revised form 31 May 2024; Accepted 1 June 2024

Available online 6 June 2024
0021-9991/© 2024 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

methods focusing on stiff matrix construction or algorithm design, from a mesh perspective, mesh-adaptive methods can be employed to resolve the locations where the regularity of solution is low.

Currently, there are three main types of mesh-adaptive methods: *h*-adaptive methods that focus on mesh refinement, *p*-adaptive methods that focus on increasing the degree of basis functions, and *r*-adaptive methods that focus on relocating mesh. The *h*-adaptive and *p*-adaptive methods both require increasing the degrees of freedom (DOFs) in the system, which may give rise to challenges [63] such as leading to excessively large number of DOFs or unnecessarily deteriorating the condition number of the linear system during the adaptive process. In practical problems, engineers can often estimate the required number of freedoms to achieve the desired accuracy based on experience and a basic upper bound on problem size. Therefore, *r*-adaptive methods, which do not need to increase the DOFs, deserve more attention.

In our study, we utilize the *r*-adaptive method, also known as the moving mesh method. This approach allows us to redistribute mesh points based on the geometric shape of the region or the error estimate of the solution. Then we are able to achieve a more even distribution of error across each element, which leads to improved solution accuracy and convergence performance. The *r*-adaptive method has two key applications. Firstly, it can optimize the initial mesh in the region with complex geometry. Secondly, during the iterative process of solution, it dynamically adjusts the mesh density based on various criteria such as residual, energy, or specially designed objective functions. The dynamic adjustment enables us to capture the locations of singularities and boundary layers while avoiding the issue of excessive growth of DOFs.

There are two primary categories of available moving mesh methods [35]: velocity-based methods and location-based methods. Velocity-based methods, such as the Geometric Conservation Law (GCL) [53] and the Moving Finite Element Method (MFEM) [50], excel in maintaining conservation properties. However, the issue of mesh entanglement resulting from rapid mesh movement has always been a challenging and significant research topic in this field. Location-based methods can be further divided into two subclasses, depending on whether a reference mesh is introduced or not. The first subclass only considers the movement of the physical mesh itself and requires the formulation of an objective function that incorporates the positions of mesh nodes as variables. Typically, during optimization, each mesh point can be moved in the opposite direction of the shape gradient to minimize the objective function [19,54]. The second subclass treats mesh movement as the change of mapping from a fixed reference mesh to the physical mesh. The objective function in this case is designed as a functional that depends on the mapping or its inverse. The optimization process usually involves solving the Euler-Lagrange equations or their corresponding gradient flow equations, such as the Moving Mesh Partial Differential Equations (MMPDE) [33,34].

The methods in the second subclass heavily rely on the choice of the objective functional and the *monitor function*. The monitor function, essentially a Riemannian metric matrix, is central in various works aiming to design and select suitable monitor functions tailored to different requirements. These efforts primarily fall into two categories: isotropic and anisotropic monitor functions. The former, isotropic monitor functions, are a natural extension of one-dimensional mesh density functions, characterized by their simplicity and effectiveness. They require determining a single variable function, ρ, and numerous existing works have adopted this form of monitor function [15–18,20,24,28,37,39,43,46,51,56,59,64]. The latter, anisotropic monitor functions, are designed within a framework that can be understood as compressing and stretching along the directions of the matrix's eigenvectors, corresponding to changes in the associated eigenvalues [10]. For two-dimensional and three-dimensional problems, this framework of anisotropic monitor functions offers greater flexibility compared to its isotropic counterparts, enabling more suitable adaptation of mesh size, shape, and orientation to specific problem characteristics. Several works have proposed such forms of monitor functions and analyzed their behavior in driving mesh movement [1,10,12,23,32].

In this work, we attempt to enhance the moving mesh method initially proposed by Li et al. [21,40], and apply it to the electro-diffusion coupled PNP equations. For a PNP system, the flux formed by charged particles is a significant physical quantity as it is usually the practically measured quantity and the conservation of it is also a desirable property in designing stable and accurate numerical algorithm [25,65,67]. Considering the importance of flux for the PNP system, we devise a novel flux-based monitor function for a more general functional tailored for the PNP equations. A series of numerical examples clearly illustrate that the moving mesh method can dynamically adjust the positions of mesh points, leading to a finite element space that offers a more accurate approximation of the solutions. Furthermore, the moving mesh finite element method we employed achieves better performance than that with the traditional monitor function or fixed mesh FEM in terms of solution accuracy. Moreover, in certain scenarios, it showcases better alignment with the intrinsic physical properties of energy dissipation in the PNP equation. These results demonstrate the effectiveness of our proposed approach.

The paper is structured as follows. In Section 2, we introduce the notations that will be consistently used throughout the text. In Section 3, we first give a brief introduction to the PNP model, followed by the numerical discretization methods for the PNP equations in time and space. In Section 4, the detailed implementation of moving mesh method is described. Particularly, a general functional and a novel flux-based monitor function are proposed, together with their qualitative explanation. In Section 5, we provide some numerical examples to demonstrate the effectiveness of the moving mesh method and the better performance of our approach. Finally, Section 6 contains some concluding remarks.

2. Preliminaries

Let $\Omega \subset \mathbb{R}^d$ (where $d = 2, 3$) denote the physical region where the PNP equations will be solved. $\partial\Omega$ is the boundary of the physical region Ω. In this paper, we adopt the standard notation for Sobolev spaces $W^{s,p}(\Omega)$, including their associated norms $\|\cdot\|_{s,p,\Omega}$ and seminorms $|\cdot|_{s,p,\Omega}$ [13]. Specifically, when $p = 2$, we use $H^s(\Omega) = W^{s,2}(\Omega)$ and $H_0^1(\Omega) = \{v \in H^1(\Omega) : v|_{\partial\Omega} = 0\}$, where the condition $v|_{\partial\Omega} = 0$ is understood in terms of trace. The norm $\|\cdot\|_{s,2,\Omega}$ is denoted as $\|\cdot\|_{s,\Omega}$. The dual of $H_0^1(\Omega)$ is denoted

as $H^{-1}(\Omega)$. Let $(\cdot,\cdot)$ be the standard inner product of $L^2(\Omega)$ and the dual inner product from $H_0^1(\Omega)$ to $H^{-1}(\Omega)$ be denoted as $\langle\cdot,\cdot\rangle$. The symbol $|\cdot|$ is used to represent the length of a vector or the magnitude of scalar.

In addition, for a time-dependent function, the proper space is denoted as $L^1(0,T;H^1(\Omega)) = \{v(\boldsymbol{x},t) \in H^1(\Omega) : \int_0^T \|v\|_{1,\Omega} \mathrm{d}t < \infty\}$. And the corresponding norm can be defined as

$$\|v\|_{L^1(0,T;H^1(\Omega))} = \int_0^T \|v\|_{1,\Omega} \mathrm{d}t, \quad \forall v \in L^1(0,T;H^1(\Omega)). \tag{1}$$

For the sake of brevity in subsequent discussions, it is necessary to provide some notations in the physical region Ω and the corresponding reference region $\Omega_C \subset \mathbb{R}^d$, which is also called *computational region*. The coordinate of the point in Ω is denoted as $\boldsymbol{x} = [x^1, x^2, \dots, x^d]^\top$ and the coordinate of the point in Ω_C is denoted as $\boldsymbol{\xi} = [\xi^1, \xi^2, \dots, \xi^d]^\top$.

We assume that the physical region Ω is a polyhedron (polygon) whose boundary can be defined as the union of facets (edges) $\Gamma_i \subset \mathbb{R}^{d-1}$, i.e., $\partial\Omega = \cup_{i=1}^{N_\Gamma} \Gamma_i$, where N_Γ is the number of boundary facets (edges). Correspondingly, the computational region Ω_C is also a polyhedron (polygon) with the same number of boundary facets (edges), and $\partial\Omega_C$ can be rewritten as $\partial\Omega_C = \cup_{i=1}^{N_\Gamma} \Gamma_i^C$, where $\Gamma_i^C \subset \mathbb{R}^{d-1}$ is the boundary facet (edge) of Ω_C.

After denoting geometric objects in the region, we need to further define meshes in the region. The mesh composed of simplices on Ω can be represented as $\mathcal{T} = \{\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}\}$. Here, $\mathcal{V}_\mathcal{T} = \{\boldsymbol{x}_i\}_{i=1}^{N_v}$ represents the collection of all mesh points in $\mathcal{T}$, where N_v denotes the total number of mesh points, encompassing both boundary and interior points. The set of interior points and boundary points can be denoted as $\mathcal{V}_\mathcal{T}^{in}$ and $\mathcal{V}_\mathcal{T}^{bd}$, respectively. Without loss of generality, we assume that the first $\tilde{N}_v$ points ($\tilde{N}_v < N_v$) are interior points, and the remaining $N_v - \tilde{N}_v$ ones belong to the boundary $\partial\Omega$, i.e., $\mathcal{V}_\mathcal{T} = \mathcal{V}_\mathcal{T}^{in} \cup \mathcal{V}_\mathcal{T}^{bd}$ where $\mathcal{V}_\mathcal{T}^{in} := \{\boldsymbol{x}_i\}_{i=1}^{\tilde{N}_v}$, $\mathcal{V}_\mathcal{T}^{bd} := \{\boldsymbol{x}_i\}_{i=\tilde{N}_v+1}^{N_v}$. The set of all elements in $\mathcal{T}$ is denoted as $\mathcal{E}_\mathcal{T} = \{\boldsymbol{e}_i\}_{i=1}^{N_e}$, where N_e is the total number of elements. Similarly, the mesh on the computational region Ω_C is represented as $\mathcal{T}_C = \{\mathcal{V}_{\mathcal{T}_C}, \mathcal{E}_{\mathcal{T}_C}\}$, where $\mathcal{V}_{\mathcal{T}_C} = \{\boldsymbol{\xi}_i\}_{i=1}^{N_v}, \mathcal{E}_{\mathcal{T}_C} = \{\omega_i\}_{i=1}^{N_e}$.

We can further define continuous piecewise polynomial spaces on the mesh $\mathcal{T}$, i.e.,

$$V_\mathcal{T}^k(\Omega) = \{v \in C(\bar{\Omega}) : v|_e \in \mathcal{P}_k(e), \forall e \in \mathcal{E}_\mathcal{T}\}, \tag{2}$$

where $\bar{\Omega}$ is the closure of Ω and $\mathcal{P}_k(e)$ denotes the space of degree-k polynomial functions defined on the element e. $V_{\mathcal{T}_C}^k(\Omega_C)$ is the corresponding space defined on the mesh $\mathcal{T}_C$. Let $\{\varphi_i\}_{i=1}^N$ be a basis of finite dimensional space $V_\mathcal{T}^k(\Omega)$, where N is the number of DOFs. We denote the corresponding interpolation points for the i-th basis function φ_i as $\boldsymbol{q}_i$. Particularly, when $k = 1$ and the interpolation points are the vertices of elements, $V_\mathcal{T}^1(\Omega)$ is the Lagrange linear element space, where the number of DOFs is equal to the number of mesh points, i.e., $N = N_v$. We denote these linear basis functions as special notation $\{\phi_i\}_{i=1}^{N_v}$.

3. Poisson–Nernst–Planck equations

In this part, we first introduce the PNP equations which consist of the Poisson equation and the Nernst-Planck equations. Subsequently, some types of boundary conditions for the PNP equations will be elucidated.

PNP equations describe a system with K species of charged particles driven by Brownian motion and the electric field. We consider the system in a bounded domain $\Omega \subset \mathbb{R}^d (d = 2, 3)$ with smooth boundary $\partial\Omega$. Then the dimensionless PNP equations can be written as

$$\begin{cases} \frac{\partial c_k}{\partial t} = -\nabla \cdot \boldsymbol{J}_k & \text{in } \Omega, \\ \boldsymbol{J}_k = -D_k(\nabla c_k + z_k c_k \nabla \Phi) & k = 1, \dots, K, \\ -\nabla \cdot (\epsilon \nabla \Phi) = \rho_0 + \sum_{k=1}^K z_k c_k & \text{in } \Omega, \end{cases} \tag{3}$$

where some dimensionless quantities are defined as follows.

- $c_k(\boldsymbol{x}, t)$ is the density of the k-th species.
- $\Phi(\boldsymbol{x}, t)$ is the electric potential contributed by the charged particles.
- $\rho_0(\boldsymbol{x})$ is a given fixed charge density function.
- z_k, D_k are the valence and the diffusion constant of the k-th species, which can be set as constants for simplicity. Here, D_k should always be positive.
- $\epsilon > 0$ is the dielectric constant.
- $\boldsymbol{J}_k$ is the flux formed by the k-th species, which can be rewritten as $\boldsymbol{J}_k = c_k \boldsymbol{v}_k = -D_k c_k \nabla \mu_k$. Here, $\boldsymbol{v}_k$ and μ_k are the velocity field and chemical potential of the k-th species. Clearly, $\mu_k = \log c_k + z_k \Phi$.

The initial value of c_k should be given, such as

$$c_k(\boldsymbol{x}, 0) = c_{k,0}, \quad k = 1, 2, \dots, K. \tag{4}$$

The boundary conditions imposed on Φ can usually be Dirichlet or Neumann in the physical sense, which are as follows.

$$\Phi|_{\partial\Omega} = V(\boldsymbol{x},t), \quad \epsilon\frac{\partial\Phi}{\partial\boldsymbol{n}}\Big|_{\partial\Omega} = \sigma_s(\boldsymbol{x},t), \tag{5}$$

where $\boldsymbol{n}$ is outer normal at the boundary of Ω, $V(\boldsymbol{x},t)$ is imposed voltages, and $\sigma_s(\boldsymbol{x},t)$ is a given surface charge density on the boundary. For the concentrations c_k, there are two kinds of boundary conditions. One is imposed on the concentrations c_k directly, which is called *Dirichlet boundary conditions*. The other is imposed on the fluxes $\boldsymbol{J}_k$, which is called *blocking boundary conditions*. Both of them are as follows.

$$c_k|_{\partial\Omega} = \gamma_k(\boldsymbol{x},t), \quad \boldsymbol{J}_k\cdot\boldsymbol{n}|_{\partial\Omega}(\boldsymbol{x},t) = 0, \quad k = 1,2,\dots,K, \tag{6}$$

where $\gamma_k(\boldsymbol{x},t)$ is the given concentration on the boundary.

Besides, the free energy of the PNP model is defined as follows without considering the boundary conditions.

$$E[c_1,c_2,\dots,c_K] = \int_\Omega \Big(\sum_{k=1}^K c_k\log c_k + \frac{1}{2}(\rho_0+\sum_{k=0}^K z_k c_k)\Phi\Big)\mathrm{d}\boldsymbol{x}. \tag{7}$$

Then we can easily obtain the property of energy dissipation with the following derivation.

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \int_\Omega \sum_{k=1}^K \frac{\delta E}{\delta c_k}\frac{\partial c_k}{\partial t}\mathrm{d}\boldsymbol{x} = -\int_\Omega \sum_{k=1}^K D_k c_k|\nabla\mu_k|^2\mathrm{d}\boldsymbol{x} \le 0. \tag{8}$$

For a general and rigorous free energy formulation and discussion, please refer to [44].

4. Numerical discretization

We consider an implicit discretization of (3) with the selective boundary condition. Firstly, we need to designate some notations. Let $\Gamma_{D,k}\subset\partial\Omega$ be the boundary part with Dirichlet boundary condition for c_k and $\Gamma_{D,\Phi}\subset\partial\Omega$ be the boundary part with Dirichlet boundary condition for Φ. We can denote the test finite element space $V_{c_k}^p = \{v\in V_{\mathcal{T}}^p(\Omega):\ v|_{\Gamma_{D,k}} = 0\}, V_\Phi^p = \{v\in V_{\mathcal{T}}^p(\Omega):\ v|_{\Gamma_{D,\Phi}} = 0\}$ ($p\ge 1$). Given a partition on the time interval $[0,T]$, i.e., $0 = t_0 < t_1 < \cdots < t_M = T$, any function $f(\boldsymbol{x},t)$ at time t_m can be denoted as $f^{(m)}$.

Then the specific scheme of the implicit discretization is given as follows.

Scheme 4.1. *With appropriate initial status* $(c_1^{(0)},c_2^{(0)},\dots,c_K^{(0)})\in[V_{\mathcal{T}}^p(\Omega)]^K$, *find* $(c_1^{(m)},c_2^{(m)},\dots,c_K^{(m)},\Phi^{(m)})\in[V_{\mathcal{T}}^p(\Omega)]^K$ *which meets all Dirichlet boundary conditions for every* $m\ge 1$, *such that for all* $(v_0,v_1,\dots,v_K)\in V_\Phi^p\times V_{c_1}^p\times\cdots\times V_{c_K}^p$ *holds*

$$\begin{cases} \displaystyle\int_\Omega \frac{c_k^{(m)}-c_k^{(m-1)}}{\delta t}v_k\mathrm{d}\boldsymbol{x} + \int_\Omega D_k(\nabla c_k^{(m)} + z_k c_k^{(m)}\nabla\Phi^{(m)})\cdot\nabla v_k\mathrm{d}\boldsymbol{x} = 0 \quad k = 1,2,\dots,K, \\ \displaystyle\int_\Omega \epsilon\nabla\Phi^{(m)}\cdot\nabla v_0\mathrm{d}\boldsymbol{x} = \int_\Omega \rho_0 v_0\mathrm{d}\boldsymbol{x} + \sum_{k=1}^K\int_\Omega z_k c_k^{(m)}v_0\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega\setminus\Gamma_{D,\Phi}}\sigma_s^{(m)}v_0\mathrm{d}\boldsymbol{x}, \end{cases} \tag{9}$$

where δt *is the time step and* σ_s *is given by the boundary condition* $\epsilon\frac{\partial\Phi}{\partial\boldsymbol{n}}|_{\partial\Omega\setminus\Gamma_{D,\Phi}} = \sigma_s$.

To avoid solving the coupled system directly in (9), we adopt Gummel iteration to decouple it. This decoupling method is a kind of nonlinear block Gauss-Seidel iteration. If we abstract the equations in (9) as

$$\begin{cases} F_k(\Phi^{(m)},c_k^{(m)}) = 0 \quad k = 1,2,\dots,K, \\ F_0(\Phi^{(m)},c_1^{(m)},c_2^{(m)},\dots,c_K^{(m)}) = 0. \end{cases} \tag{10}$$

Then the Gummel iteration can be specifically expressed as Algorithm 1.

5. Moving mesh method

In this section, we provide a comprehensive and detailed introduction to the moving mesh method. We first give Algorithm 2 as an overview of the algorithm. Then three key components of the algorithm are discussed in the following subsections, namely the mesh redistribution strategy, monitor function design, and solution interpolation.

The relationship between the computational domain Ω_C and the physical domain Ω, as mentioned in the algorithm, is visually depicted in Fig. 1 (a). Additionally, to bolster comprehension of the mesh redistribution strategy discussed in Section 5.1, we present a simplified example of the mesh movement process with a solitary interior mesh point in Fig. 1 (b).

Algorithm 1 Gummel iteration for the PNP model.

1: For $m \geq 1$, set $\Phi^{m,0} = \Phi^{(m-1)}, c_k^{m,0} = c_k^{(m-1)}, k = 1, 2, \ldots, K$ and $l = 0$.
2: For $l \geq 1$, compute $\Phi^{m,l}, c_k^{m,l}, k = 1, 2, \ldots, K$ such that all

$$\begin{cases} F_k(\Phi^{m,l-1}, c_k^{m,l}) = 0 \quad k = 1, 2, \ldots, K, \\ F_0(\Phi^{m,l}, c_1^{m,l}, c_2^{m,l}, \ldots, c_K^{m,l}) = 0. \end{cases} \tag{11}$$

3: For fixed tol > 0, stop if

$$\|\Phi^{m,l} - \Phi^{m,l-1}\|_{0,\Omega} < \text{tol}\,, \quad \|c_k^{m,l} - c_k^{m,l-1}\|_{0,\Omega} < \text{tol}\,, \; k = 1, 2 \ldots, K. \tag{12}$$

Set $\Phi^{(m)} \leftarrow \Phi^{m,l}, c_k^{(m)} \leftarrow c_k^{m,l}, k = 1, 2, \ldots, K$ and go to step 4; otherwise, set $l \leftarrow l + 1$ and go to step 2.
4: Set $m \leftarrow m + 1$ and go to step 1.

Algorithm 2 Moving mesh method.

Require: A fixed computational mesh $\mathcal{T}_C^*$ (see Section 5.4.1), along with the solution of the PNP equations at time $t^{(m)}$, represented by $\Phi^{(m)}$, $c_1^{(m)}$, …, $c_K^{(m)}$, and a specified tolerance value, referred to as tol.
1: Obtain the mesh $\mathcal{T}_C$ in computational region Ω_C by solving the optimization problem (see Section 5.1) with respect to the variables $\Phi^{(m)}, c_1^{(m)}, \ldots, c_K^{(m)}$.
2: **while** $\|\mathcal{T}_C - \mathcal{T}_C^*\|_{L^\infty(\Omega_C)} >$ tol **do**
3: Use the difference between $\mathcal{T}_C$ and $\mathcal{T}_C^*$ to guide the movement of mesh points in the current mesh $\mathcal{T}$ within the physical region Ω (see Section 5.1).
4: Interpolate the solution $\Phi^{(m)}, c_1^{(m)}, \ldots, c_K^{(m)}$ from the old mesh to the new mesh (see Section 5.2).
5: Solve the optimization problem (see Section 5.1) with respect to the updated variables $\Phi^{(m)}$, $c_1^{(m)}$, …, $c_K^{(m)}$ to get $\mathcal{T}_C$ in Ω_C.
6: **end while**

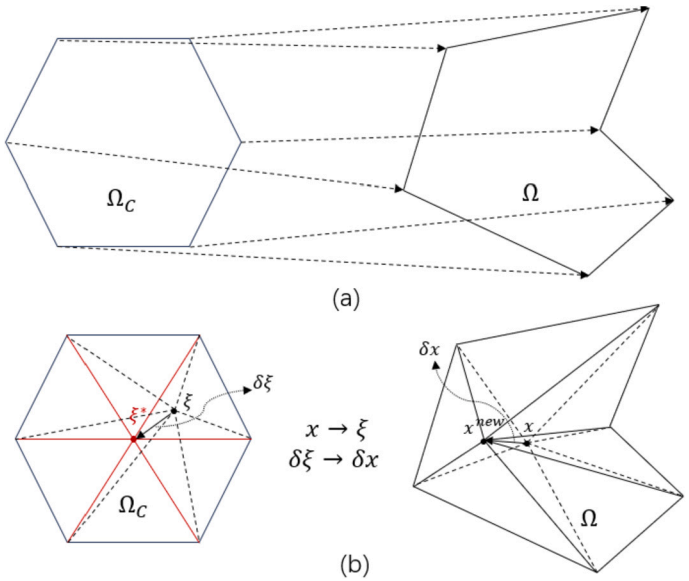


Fig. 1. (a) The relationship between the computational domain Ω_C and the physical domain Ω. (b) A simplified example of the mesh movement process with a solitary interior mesh point.

5.1. Mesh redistribution strategy

The mesh redistribution strategy aims to reposition mesh points to achieve an even redistribution of errors or user-specified quantities within each mesh element. We need to achieve the goal mathematically via solving an optimization problem or a PDE model. In this work, we use the second class of location-based methods mentioned in the introduction. It is worth highlighting that in this subsection, we present our derivation in matrix form, enhancing the clarity of the process and ensuring its applicability to any given dimension d.

We define a computational region $\Omega_C \in \mathbb{R}^d$ and a reference mesh $\mathcal{T}_C^* = \{\mathcal{V}_{\mathcal{T}_C^*} = \{\boldsymbol{\xi}_i^*\}_{i=1}^{N_v}, \mathcal{E}_{\mathcal{T}_C^*} = \{\omega_i^*\}_{i=1}^{N_e}\}$ on it. Then any mesh $\mathcal{T}$ in the physical region $\Omega \in \mathbb{R}^d$ can be described as the image of $\mathcal{T}_C^*$ under a reversible mapping function $\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{\xi}) : \Omega_C \to \Omega$, which is called an *adaptive mesh generation function*. Consequently, the optimization problem can be formulated by optimizing an objective functional of either the mapping $\boldsymbol{x}(\boldsymbol{\xi})$ or its inverse mapping $\boldsymbol{\xi}(\boldsymbol{x}) : \Omega \to \Omega_C$. However, it has been shown in [24] that optimizing the objective functional of $\boldsymbol{x}(\boldsymbol{\xi})$ may result in mesh folding and tangling more easily, especially when dealing with concave physical regions. Therefore, it is generally preferred to take the functional of $\boldsymbol{\xi}(\boldsymbol{x})$ as the objective.

In existing methods for mesh generation and adaptation, a general functional of $\boldsymbol{\xi}(\boldsymbol{x})$ can be formulated as follows [6,34,39,58]

$$G[\boldsymbol{\xi}] = \frac{1}{2}\sum_{k=1}^{d}\int_{\Omega}(\nabla\xi^k)^{\top}\boldsymbol{M}^{-1}\nabla\xi^k \mathrm{d}\boldsymbol{x}, \tag{13}$$

where $\boldsymbol{M}\in\mathbb{R}^{d\times d}$ is a symmetric positive definite matrix, which is known as *monitor function*. For this adaptation functional, a qualitative explanation of the relationship between the monitor function and mesh movement has already been provided in [10]. It should be noted that the functional derived from the harmonic mapping in [21,40] is a special case of (13) with $\boldsymbol{M} = \tilde{\boldsymbol{M}}/\det(\tilde{\boldsymbol{M}})$, where $\tilde{\boldsymbol{M}}\in\mathbb{R}^{d\times d}$ is a given metric tensor for Ω.

During optimization, it is necessary to keep the boundary of the physical region invariant. For this purpose, we can enforce that the vertices, edges, and faces of the physical region's boundary are mapped to their corresponding counterparts in the computational region, i.e., $\boldsymbol{\xi}(\Gamma_i) = \Gamma_i^C, i = 1,2,\dots,N_\Gamma$. This implies that the set for $\boldsymbol{\xi}_b(\boldsymbol{x}) = \boldsymbol{\xi}|_{\partial\Omega}$ is given by

$$\mathcal{B} = \{\boldsymbol{\xi_b}\in C(\partial\Omega):\ \boldsymbol{\xi_b}:\partial\Omega\to\partial\Omega_C; \boldsymbol{\xi_b}|_{\Gamma_i} \text{ is a linear segment and strictly increasing}, i = 1,2,\dots,N_\Gamma\}. \tag{14}$$

From some perspective, the constraint in equation (14) is overly restrictive, as it prohibits points from moving between adjacent boundary facets (edges). This may result in some distortion of mesh near the boundary during movement, particularly in practical cases with complex boundaries.

In summary, the complete optimization problem for mesh redistribution can be formulated as

$$\begin{aligned}&\min_{\boldsymbol{\xi}} G[\boldsymbol{\xi}] = \frac{1}{2}\sum_{k=1}^{d}\int_{\Omega}(\nabla\xi^k)^{\top}\boldsymbol{M}^{-1}\nabla\xi^k \mathrm{d}\boldsymbol{x},\\ &\text{s.t. } \boldsymbol{\xi}|_{\partial\Omega} = \boldsymbol{\xi}_b\in\mathcal{B}.\end{aligned} \tag{15}$$

To discrete (15), Lagrange linear finite element space $V_{\mathcal{T}}^1(\Omega)$ is used. Then $\xi^k(\boldsymbol{x})$ can be approximated as $\sum_{j=1}^{N_v}\xi_j^k\phi_j(\boldsymbol{x}) = (\boldsymbol{\xi}^k)^{\top}\boldsymbol{\phi}(\boldsymbol{x})$, where $\boldsymbol{\xi}^k = [\xi_1^k,\xi_2^k,\dots,\xi_{N_v}^k]^{\top}$, $\boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}),\phi_2(\boldsymbol{x}),\dots,\phi_{N_v}(\boldsymbol{x})]^{\top}$. Therefore, the approximation for $\boldsymbol{\xi}$ in $[V_{\mathcal{T}}^1(\Omega)]^d$ is

$$\boldsymbol{\xi}(\boldsymbol{x}) = \begin{bmatrix}\xi^1(\boldsymbol{x})\\ \xi^2(\boldsymbol{x})\\ \vdots\\ \xi^d(\boldsymbol{x})\end{bmatrix} = \begin{bmatrix}\sum_j\xi_j^1\phi_j(\boldsymbol{x})\\ \sum_j\xi_j^2\phi_j(\boldsymbol{x})\\ \vdots\\ \sum_j\xi_j^d\phi_j(\boldsymbol{x})\end{bmatrix} = \begin{bmatrix}(\boldsymbol{\xi}^1)^{\top}\\ (\boldsymbol{\xi}^2)^{\top}\\ \vdots\\ (\boldsymbol{\xi}^d)^{\top}\end{bmatrix}\boldsymbol{\phi}(\boldsymbol{x}). \tag{16}$$

Substituting (16) into the functional $G[\boldsymbol{\xi}]$ in (15), we obtain

$$\begin{aligned}G[\boldsymbol{\xi}] &= \frac{1}{2}\sum_{k=1}^{d}\int_{\Omega}[\nabla(\sum_{i=1}^{N_v}\xi_i^k\phi_i)]^{\top}\boldsymbol{M}^{-1}[\nabla(\sum_{j=1}^{N_v}\xi_j^k\phi_j)]\mathrm{d}\boldsymbol{x}\\ &= \frac{1}{2}\sum_{k=1}^{d}\sum_{i,j=1}^{N_v}\xi_i^k\xi_j^k\Big[\int_{\Omega}(\nabla\phi_i)^{\top}\boldsymbol{M}^{-1}\nabla\phi_j\mathrm{d}\boldsymbol{x}\Big]\\ &= \frac{1}{2}\sum_{k=1}^{d}(\boldsymbol{\xi}^k)^{\top}\boldsymbol{H}\boldsymbol{\xi}^k,\end{aligned} \tag{17}$$

where the entry of $\boldsymbol{H}\in\mathbb{R}^{N_v\times N_v}$ is $H_{i,j} = \int_{\Omega}(\nabla\phi_i)^{\top}\boldsymbol{M}^{-1}\nabla\phi_j\mathrm{d}\boldsymbol{x}$.

Besides, the boundary condition $\boldsymbol{\xi}|_{\partial\Omega} = \boldsymbol{\xi}_b$ can be rewritten as

$$\boldsymbol{\xi}\cdot\boldsymbol{n} = b, \tag{18}$$

where $\boldsymbol{n} = \boldsymbol{n}(\boldsymbol{x}) = [n^1(\boldsymbol{x}),n^2(\boldsymbol{x}),\dots,n^d(\boldsymbol{x})]^{\top}\in\mathbb{R}^d$ is the unit normal vector of the boundary point and $b = b(\boldsymbol{x})$ is a given function related to the boundary of the computational region $\partial\Omega_C$. It should be noted that $\boldsymbol{n}, b$ remains constant on each boundary face(edge) Γ_i. Since there are no normal vectors defined for interior points of Ω, we can assume that $\boldsymbol{n}(\boldsymbol{x}) = \boldsymbol{0}$ for any interior points. This enables us to construct a matrix $\boldsymbol{N}\in\mathbb{R}^{d\times N_v}$ by arranging $\boldsymbol{n}(\boldsymbol{x})$ corresponding to all mesh points $\boldsymbol{x}\in\mathcal{V}_{\mathcal{T}}$ as column vectors, which is as follows

$$\boldsymbol{N} = \begin{bmatrix}\boldsymbol{n}(\boldsymbol{x}_1) & \boldsymbol{n}(\boldsymbol{x}_2) & \cdots & \boldsymbol{n}(\boldsymbol{x}_{N_v})\end{bmatrix} = \begin{bmatrix}0 & \cdots & 0 & n^1_{\tilde{N}_v+1} & n^1_{\tilde{N}_v+2} & \cdots & n^1_{N_v}\\ 0 & \cdots & 0 & n^2_{\tilde{N}_v+1} & n^2_{\tilde{N}_v+2} & \cdots & n^2_{N_v}\\ \vdots & & \vdots & \vdots & \vdots & & \vdots\\ 0 & \cdots & 0 & n^d_{\tilde{N}_v+1} & n^d_{\tilde{N}_v+2} & \cdots & n^d_{N_v}\end{bmatrix}, \tag{19}$$

where the j-th column of $\boldsymbol{N}$ is $\boldsymbol{n}(\boldsymbol{x}_j) = [n_j^1,n_j^2,\dots,n_j^{N_v}]^{\top}, \boldsymbol{x}_j\in\mathcal{V}_{\mathcal{T}}$. Furthermore, if we denote each row of the matrix $\boldsymbol{N}$ as $\boldsymbol{n}^k$, $k = 1,2,\dots,d$, the discrete form of (18) on mesh $\mathcal{T}_C$ can be expressed as

$$\begin{bmatrix} \boldsymbol{N}^1 & \boldsymbol{N}^2 & \cdots & \boldsymbol{N}^d \end{bmatrix} \begin{bmatrix} \xi^1 \\ \xi^2 \\ \vdots \\ \xi^d \end{bmatrix} = \boldsymbol{b}, \tag{20}$$

where $\boldsymbol{N}^k = \mathrm{diag}(\boldsymbol{n}^k) \in \mathbb{R}^{N_v \times N_v}$, $k = 1, 2, \ldots, d$ and $\boldsymbol{b} = [0, \ldots, 0, b_{\tilde{N}_v+1}, b_{\tilde{N}_v+2}, \ldots, b_{N_v}]^\top \in \mathbb{R}^{N_v}$.

With above preparation, we can obtain the corresponding Lagrangian function for the discrete form of the optimization problem (15), which is given by

$$\begin{aligned} L[\xi, \lambda] &= G[\xi] + \sum_{j=\tilde{N}_v+1}^{N_v} \lambda_j (\xi(\boldsymbol{x}_j) \cdot \boldsymbol{n}(\boldsymbol{x}_j) - b(\boldsymbol{x}_j)) \\ &= \frac{1}{2} \begin{bmatrix} (\xi^1)^\top & (\xi^2)^\top & \ldots & (\xi^d)^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{H} & & & \\ & \boldsymbol{H} & & \\ & & \ddots & \\ & & & \boldsymbol{H} \end{bmatrix} \begin{bmatrix} \xi^1 \\ \xi^2 \\ \vdots \\ \xi^d \end{bmatrix} + \lambda^\top \left(\begin{bmatrix} \boldsymbol{N}^1 & \boldsymbol{N}^2 & \cdots & \boldsymbol{N}^d \end{bmatrix} \begin{bmatrix} \xi^1 \\ \xi^2 \\ \vdots \\ \xi^d \end{bmatrix} - \boldsymbol{b} \right), \end{aligned} \tag{21}$$

where the elements of $\lambda = [0, \ldots, 0, \lambda_{\tilde{N}_v+1}, \lambda_{\tilde{N}_v+2}, \ldots, \lambda_{N_v}]^\top \in \mathbb{R}^{N_v}$ are the Lagrange multipliers. Therefore, the optimization problem (15) is equivalent to solving the linear system

$$\begin{bmatrix} \boldsymbol{H} & & & & \boldsymbol{N}^1 \\ & \boldsymbol{H} & & & \boldsymbol{N}^2 \\ & & \ddots & & \vdots \\ & & & \boldsymbol{H} & \boldsymbol{N}^d \\ \boldsymbol{N}^1 & \boldsymbol{N}^2 & \cdots & \boldsymbol{N}^d & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \xi^1 \\ \xi^2 \\ \vdots \\ \xi^d \\ \lambda \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{b} \end{bmatrix}. \tag{22}$$

By solving the optimization problem above, we can obtain a mesh $\mathcal{T}_C$ in the computational region Ω_C, that corresponds to the current physical mesh $\mathcal{T}$. If the difference between $\mathcal{T}_C$ and the given fixed mesh $\mathcal{T}_C^*$ is not small enough, we should update the mesh $\mathcal{T}$ so that the corresponding mesh in Ω_C becomes closer to $\mathcal{T}_C^*$.

Next, we need to determine the movement direction $\delta \boldsymbol{x}_j$ for each mesh point $\boldsymbol{x}_j \in \mathcal{V}_\mathcal{T}$. After solving (22), the direction of movement for each mesh point in $\mathcal{T}_C$ can be chosen as $\delta \xi_j = \xi_j^* - \xi_j$. And we can naturally obtain the direction of movement $\delta \xi$ for any point $\xi \in \Omega_C$ through linear interpolation. In fact, the topology of the physical mesh and the reference mesh remains the same if we choose appropriate moving step. Hence, each simplex element $e \in \mathcal{E}_\mathcal{T}$ has a unique corresponding simplex element $\omega \in \mathcal{E}_{\mathcal{T}_C}$. For a given element $e \in \mathcal{E}_\mathcal{T}$ and its corresponding element $\omega \in \mathcal{E}_{\mathcal{T}_C}$, we denote $\{\boldsymbol{x}_{e,i}\}_{i=0}^d$ and $\{\boldsymbol{\xi}_{\omega,i}\}_{i=0}^d$ as their vertices, respectively. Then any point inside the elements e or ω can be represented in barycentric coordinates $\boldsymbol{\alpha} = [\alpha^0, \alpha^1, \ldots, \alpha^d]^\top$ as

$$\begin{aligned} \boldsymbol{x} &= \sum_{i=0}^d \alpha^i \boldsymbol{x}_{e,i} = \boldsymbol{x}_{e,0} + \sum_{i=1}^d \alpha^i (\boldsymbol{x}_{e,i} - \boldsymbol{x}_{e,0}), \\ \boldsymbol{\xi} &= \sum_{i=0}^d \alpha^i \boldsymbol{\xi}_{\omega,i} = \boldsymbol{\xi}_{\omega,0} + \sum_{i=1}^d \alpha^i (\boldsymbol{\xi}_{\omega,i} - \boldsymbol{\xi}_{\omega,0}). \end{aligned} \tag{23}$$

According to the chain rule, we can get a constant Jacobian matrix on the element ω as

$$\left.\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}\right|_\omega = \left.\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\alpha}}\right|_e \left(\left.\frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{\alpha}}\right|_\omega \right)^{-1} = \begin{bmatrix} x_{e,1}^1 - x_{e,0}^1 & \cdots & x_{e,d}^1 - x_{e,0}^1 \\ \vdots & & \vdots \\ x_{e,1}^d - x_{e,0}^d & \cdots & x_{e,d}^d - x_{e,0}^d \end{bmatrix} \begin{bmatrix} \xi_{\omega,1}^1 - \xi_{\omega,0}^1 & \cdots & \xi_{\omega,d}^1 - \xi_{\omega,0}^1 \\ \vdots & & \vdots \\ \xi_{\omega,1}^d - \xi_{\omega,0}^d & \cdots & \xi_{\omega,d}^d - \xi_{\omega,0}^d \end{bmatrix}^{-1}. \tag{24}$$

If we take the volume of the element $e \in \mathcal{N}_i$ as the weight, where $\mathcal{N}_i \subset \mathcal{E}_\mathcal{T}$ is the set of elements with $\boldsymbol{x}_i$ as a vertex, the weighted average moving direction of $\boldsymbol{x}_i$ is defined by

$$\delta \boldsymbol{x}_i = \sum_{e \in \mathcal{N}_i} \left(\frac{|e|}{\sum_{e \in \mathcal{N}_i} |e|} \right) \delta \boldsymbol{x}_i \Big|_e = \sum_{e \in \mathcal{N}_i} \left(\frac{|e|}{\sum_{e \in \mathcal{N}_i} |e|} \right) \left.\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}\right|_\omega \delta \boldsymbol{\xi}_i. \tag{25}$$

Therefore, with a given update step size of θ, the physical mesh can be updated as

$$\boldsymbol{x}_i^{\mathrm{new}} = \boldsymbol{x}_i^{\mathrm{old}} + \theta \delta \boldsymbol{x}_i, \qquad i = 1, 2, \ldots, N_v, \tag{26}$$

where the selection of θ will be introduced in Section 5.4.2.

5.2. Solution interpolation

After the physical mesh is redistributed, it is necessary to interpolate the solution from the old mesh to the new one. The primary challenge of interpolation generally lies in finding the elements that contain the new mesh points [35]. To avoid this difficulty, a

PDE-based interpolation method proposed in [40] is adopted here. However, it is crucial to note that, distinct from the presentation in [40], we differentiate basis functions of FEM used in solving the PDEs, denoted as $\{\varphi_i\}_{i=1}^{N}$, from linear nodal basis functions, denoted as $\{\phi_i\}_{i=1}^{N_v}$. This indicates that the moving mesh technique introduced in this work is not limited to the linear nodal finite element during solving the physical PDEs, which reflects the universality of the moving mesh method.

In this subsection, the solution to be interpolated is denoted as $u(\boldsymbol{x})$. We can view the movement of physical mesh as a continuous process, i.e.,

$$\boldsymbol{x}_i(\tau) = (1-\tau)\boldsymbol{x}_i^{\text{old}} + \tau \boldsymbol{x}_i^{\text{new}}, \quad i = 1, 2, \dots, N_v, \quad \tau \in [0, 1], \tag{27}$$

where τ is a pseudo-time introduced to describe mesh movement. Obviously, $\boldsymbol{x}_i(0) = \boldsymbol{x}_i^{\text{old}}, \boldsymbol{x}_i(1) = \boldsymbol{x}_i^{\text{new}}$. Then the discrete form of u in $V_{\mathcal{T}}^k(\Omega)$ can be viewed as

$$u_h(\boldsymbol{x}, \tau) = \sum_{j=1}^{N} u_j(\tau)\varphi_j(\boldsymbol{x}, \tau). \tag{28}$$

Therefore, the interpolation of u_h from the old mesh to the new one is equivalent to finding the solution $u_h(\boldsymbol{x}, \tau)$ which obeys the following differential equation

$$\frac{\partial u_h}{\partial \tau} = 0. \tag{29}$$

According to the relationship between material and local derivative, we can get that

$$\frac{\partial \varphi_j(\boldsymbol{x}, \tau)}{\partial \tau} = -\nabla \varphi_j(\boldsymbol{x}, \tau) \cdot \delta \boldsymbol{x}, \qquad j = 1, 2, \dots, N, \tag{30}$$

where $\delta \boldsymbol{x} = \sum_{i=1}^{N_v} \phi_i(\boldsymbol{x}, \tau) \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\tau} = \sum_{i=1}^{N_v} \phi_i(\boldsymbol{x}, \tau)(\boldsymbol{x}_i^{\text{new}} - \boldsymbol{x}_i^{\text{old}})$. Here, $\phi_i \in V_{\mathcal{T}}^1(\Omega)$ is the i-th Lagrange linear basis function. The detailed derivation of (30) will be provided in the Appendix A.

With the relationship in (30) and (28), the differential equation (29) gives

$$0 = \frac{\partial u_h}{\partial \tau} = \sum_{j=1}^{N} \left(\frac{\mathrm{d}u_j}{\mathrm{d}\tau} \varphi_j + u_j \frac{\partial \varphi_j}{\partial \tau} \right) = \sum_{j=1}^{N} \frac{\mathrm{d}u_j}{\mathrm{d}\tau} \varphi_j - \sum_{j=1}^{N} u_j \nabla \varphi_j \cdot \delta \boldsymbol{x}. \tag{31}$$

As $\nabla u_h = \sum_{j=0}^{N} u_j \nabla \varphi_j$, we can obtain from the above equation that

$$\sum_{j=1}^{N} \frac{\mathrm{d}u_j}{\mathrm{d}\tau} \varphi_j(\boldsymbol{x}, \tau) - \nabla u_h \cdot \delta \boldsymbol{x} = 0. \tag{32}$$

Then the corresponding weak form of (32) can be represented as

$$\int_{\Omega} \left(\sum_{j=1}^{N} \frac{\mathrm{d}u_j}{\mathrm{d}\tau} \varphi_j(\boldsymbol{x}, \tau) - \nabla u_h \cdot \delta \boldsymbol{x} \right) v \mathrm{d}\boldsymbol{x} = 0, \qquad \forall v \in V_h(\tau). \tag{33}$$

If we let v be the basis function of $V_{\mathcal{T}}^k(\Omega)$, i.e., $v = \varphi_i(\boldsymbol{x}, \tau)$, a system of linear ODEs for u_j can be obtained, which is given by

$$\sum_{j=1}^{N} \left(\int_{\Omega} \varphi_i \varphi_j \mathrm{d}\boldsymbol{x} \right) \frac{\mathrm{d}u_j}{\mathrm{d}\tau} = \sum_{j=1}^{N} \left(\int_{\Omega} \varphi_i \nabla \varphi_j \cdot \delta \boldsymbol{x} \mathrm{d}\boldsymbol{x} \right) u_j, \qquad i = 1, 2, \dots, N. \tag{34}$$

5.3. Monitor functions

In the process of adaptive mesh movement, the effectiveness is largely determined by the monitor function $\boldsymbol{M}$. The commonly used monitor function is a diagonal matrix based on the magnitudes of gradients of all solutions in the system, given by

$$\boldsymbol{M}(\boldsymbol{x}) = \left(\sqrt{\delta + \sum_i |\nabla u_i|^2} \right) \boldsymbol{I}, \tag{35}$$

where $\{u_i\}$ represents the solutions and $\delta > 0$ is a parameter controlling the degree of mesh movement and preventing $\boldsymbol{M}$ from becoming a singular matrix. For PNP equations (3), (35) can be rewritten as

$$\boldsymbol{M}_1(\boldsymbol{x}) = \left(\sqrt{\delta + |\nabla \Phi|^2 + \sum_{k=1}^{K} \eta_k |\nabla c_k|^2} \right) \boldsymbol{I}, \tag{36}$$

where $\eta_k, k = 1, 2, \dots, K$ are parameters for regulating the impact of ion concentrations on the mesh movement.

Inspired by [10], another kind of monitor function can be designed to make the mesh move along the certain direction $\boldsymbol{v}(\boldsymbol{x})$,

$$\boldsymbol{M}(\boldsymbol{x}) = \lambda_1(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{x})\boldsymbol{v}^{\top}(\boldsymbol{x}) + \lambda_2(\boldsymbol{x})\boldsymbol{v}_{\perp}(\boldsymbol{x})\boldsymbol{v}_{\perp}^{\top}(\boldsymbol{x}), \tag{37}$$

where $\boldsymbol{v}(\boldsymbol{x}), \boldsymbol{v}_{\perp}(\boldsymbol{x})$ are normalized orthogonal eigenvectors corresponding to the positive eigenvalues $\lambda_1(\boldsymbol{x}), \lambda_2(\boldsymbol{x})$. Here, $\lambda_1(\boldsymbol{x})$ ought to pinpoint locations with local singularities or boundary layers, i.e., $\lambda_1(\boldsymbol{x})$ should have bumps along the $\boldsymbol{v}(\boldsymbol{x})$ at these locations. Besides, $\lambda_2(\boldsymbol{x})$ can be chosen as a function of $\lambda_1(\boldsymbol{x})$. In fact, the monitor function defined in (35) can be interpreted as a special case of (37) with $\lambda_1(\boldsymbol{x}) = \lambda_2(\boldsymbol{x}) = \sqrt{\delta + \sum_i |\nabla u_i|^2}$.

From a physical perspective, the flux formed by charge carriers in the PNP system is an important physical quantity, and the convection-dominant effect is closely related to the characteristics of the flux. According to lots of work and our research experiences [5,25,30,36,65,67], preserving flux conservation in numerical formulations significantly enhances algorithm stability and accuracy. While this work does not focus on the design of flux-conserving algorithms, we propose a novel monitor function based on flux for moving mesh method. This allows the mesh to contract and expand along the direction of streamlines in response to changes in the flux divergence. As a result, the moving mesh method can effectively capture the characteristics of flux in the PNP system. Specifically, $\hat{\boldsymbol{J}}_k = \frac{\boldsymbol{J}_k}{|\boldsymbol{J}_k|}$ is selected as $\boldsymbol{v}(\boldsymbol{x})$ and the divergence of flux $\nabla \cdot \boldsymbol{J}_k$ is chosen to determine the eigenvalue $\lambda_1(\boldsymbol{x})$. Then the new monitor function for the PNP equation is written as

$$\boldsymbol{M}_2(\boldsymbol{x}) = \sum_{k=1}^{K} (\lambda_1 \hat{\boldsymbol{J}}_k \hat{\boldsymbol{J}}_k^{\top} + \lambda_2 \hat{\boldsymbol{J}}_{k,\perp} \hat{\boldsymbol{J}}_{k,\perp}^{\top}), \tag{38}$$

where $\lambda_1 = \sqrt{\delta + |\nabla \cdot \boldsymbol{J}_k|^2}$ $(\delta > 0)$ and λ_2 is a positive function of λ_1. $\hat{\boldsymbol{J}}_k$ is the normalized flux vector of $\boldsymbol{J}_k$ and $\hat{\boldsymbol{J}}_{k,\perp}$ is the normalized orthogonal vector of $\hat{\boldsymbol{J}}_k$. In the following numerical experiments, we choose $\lambda_2 = 0.5\lambda_1$.

Overall, the new monitor function $\boldsymbol{M}_2$, inspired by the flux in the PNP system, is heuristically devised. Mathematically, it cannot be guaranteed that $\boldsymbol{M}_2$ consistently outperforms the traditional monitor function $\boldsymbol{M}_1$ in all scenarios. However, given that the design framework of the monitor function (37) we employ is more general, where the traditional monitor function $\boldsymbol{M}_1$ is just a specific instance within this framework, we have the flexibility to adjust parameters or devise other monitor functions tailored to specific problems. Thus, the outcomes of these alternatives are expected to be at least on par with $\boldsymbol{M}_1$.

5.4. Some details

Regarding the moving mesh method, there are a couple of details that need to be clarified. One of them is the process of creating a fixed reference mesh, and the other is to determine the appropriate step size, denoted as θ.

5.4.1. Fixed reference mesh

The first step in the moving mesh method is to determine a fixed reference mesh $\mathcal{T}_C^*$ in Ω_C. In general, the reference mesh $\mathcal{T}_C^*$ should meet certain conditions: the mesh should not be tangled with each other, and its topology should match that of the physical mesh $\mathcal{T}$. Therefore, we can solve the weak form of the following Laplace equations to determine the initial mesh positions [40,58],

$$-\nabla^2 \boldsymbol{\xi}(\boldsymbol{x}) = 0. \tag{39}$$

Additionally, some boundary conditions should be given to ensure that the reference mesh $\mathcal{T}_C^*$ and the physical mesh $\mathcal{T}$ have the same boundary structure, i.e., corresponding boundary points have the same barycentric coordinates:

$$\boldsymbol{\xi}(\boldsymbol{x}) = \sum_{j=0}^{d-1} \alpha^j(\boldsymbol{x}) \mathbf{v}_j(\Gamma_i^C), \qquad \forall \boldsymbol{x} = \sum_{j=0}^{d-1} \alpha^j(\boldsymbol{x}) \mathbf{v}_j(\Gamma_i) \in \Gamma_i \subset \partial\Omega. \tag{40}$$

Here, $\{\mathbf{v}_j(\Gamma_i^C)\}_{j=0}^{d-1}, \{\mathbf{v}_j(\Gamma_i)\}_{j=0}^{d-1}$ are the sets of vertices for Γ_i^C and Γ_i, respectively. The barycentric coordinate of $\boldsymbol{x}$ on the boundary facet(edge) is written as $\boldsymbol{\alpha} = [\alpha^0(\boldsymbol{x}), \alpha^1(\boldsymbol{x}), \ldots, \alpha^{d-1}(\boldsymbol{x})]^{\top}$.

5.4.2. Step size

When considering a given step size for mesh moving, denoted as θ, we need to avoid mesh tangle, which means maintaining the orientation of all mesh elements. Therefore, for any mesh element $e \in \mathcal{T}_h$, we need to ensure that step size $\theta > 0$ is smaller than the minimum positive solution θ^* of the following equation.

$$\det\left(\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \boldsymbol{x}_{e,0} + \theta\delta\boldsymbol{x}_{e,0} & \boldsymbol{x}_{e,1} + \theta\delta\boldsymbol{x}_{e,1} & \cdots & \boldsymbol{x}_{e,d} + \theta\delta\boldsymbol{x}_{e,d} \end{bmatrix} \right) = 0, \quad \forall e \in \mathcal{T}, \tag{41}$$

i.e., $\theta = \eta \min_{e\in\mathcal{T}} \theta^*$, where $\eta \in [0, 1)$ is a scaling factor.

Additionally, to avoid producing too small elements during mesh movement, we propose to adaptively adjust the scaling factor η. Before explaining the adaptive step size adjustment strategy, we introduce some notations.

- $\Delta_{\max}^{(l)} := \max_{e\in\mathcal{T}}\{(\max_i |\delta \boldsymbol{x}_{e,i}|)/h_e\}$, where h_e is the minimum distance from the vertices of element e to the hyperplanes that opposite them. Here, the superscript (l) denotes the interior iteration step of mesh movement.
- N_{down} denotes the number of consecutive decreases of $\Delta_{\max}$. If an increase occurs, N_{down} is reset to 0.

Then our algorithm for adaptive step size adjustment is as follows.

Algorithm 3 Adaptive step size adjustment.

Require: Given the initial value η_0 and some constants $\eta_{\min} < \hat{\eta} < \eta_{\max}, \Delta_1 < \Delta_2 < \Delta_3, N_{\text{down}}^1 < N_{\text{down}}^2$, the following algorithm is to determine the value of scaling factor at the l-th interior iteration, which is denoted as $\eta^{(l)}$.

$\eta^{(l)} \leftarrow \eta^{(l-1)}$
if $\Delta_{\max}^{(l)} > \Delta_3$ and $\eta^{(l)} > \hat{\eta}$ **then**
  Set $\eta^{(l)} \leftarrow \hat{\eta}$
end if
if $\Delta_{\max}^{(l)} > \Delta_{\max}^{(l-1)}$ and $\eta^{(l)} < \eta_{\max}$ **then**
  Set $\eta^{(l)} \leftarrow \frac{1}{2}\eta^{(l)}$ and $N_{down} \leftarrow 0$.
else
  if $\Delta_{\max}^{(l)} < \Delta_1$, $N_{\text{down}} > N_{\text{down}}^1$ and $\eta^{(l)} < \eta_{\max}$ **then**
    Set $\eta^{(l)} \leftarrow 2\eta^{(l)}$.
  else if $\Delta_{\max}^{(l)} < \Delta_2$, $N_{\text{down}} > N_{\text{down}}^2$ and $\eta^{(l)} < \eta_{\max}$ **then**
    Set $\eta^{(l)} \leftarrow 2\eta^{(l)}$.
  end if
  Set $N_{\text{down}} \leftarrow N_{\text{down}} + 1$.
end if

In practice, the constants mentioned in Algorithm 3 can be adjusted according to the specific problem. In our numerical experiments, we choose $\eta_{\min} = 0.0125, \eta_{\max} = 0.5, \hat{\eta} = 0.125, \Delta_1 = 1, \Delta_2 = 10, \Delta_3 = 20, N_{\text{down}}^1 = 10, N_{\text{down}}^2 = 20$.

6. Numerical results

6.1. Example 1

We consider the time-dependent PNP equations with a fixed singularity. The equations are as follows

$$\begin{cases} \dfrac{\partial c_k}{\partial t} - \nabla\cdot(D_k(\nabla c_k + z_k c_k \nabla\Phi)) = f_k, \quad k=1,2, \text{ in } \Omega, t\in[0,T], \\ -\nabla\cdot(\epsilon\nabla\Phi) = \displaystyle\sum_{k=1}^{2} z_k c_k + f_0, \qquad\qquad \text{in } \Omega, t\in[0,T], \end{cases} \tag{42}$$

where the computational L-shaped domain $\Omega = [-1,1]^2\backslash(-1.0\times10^{-4},1]\times(-1.0\times10^{-4},1]\in\mathbb{R}^2$ and the time span $T = 0.5$. The basic parameters for the PNP equations are set as $\epsilon = 1.0, D_1 = D_2 = 1, z_1 = 1, z_2 = -1$. The boundary condition and the source terms $f_k, k = 0,1,2$ should be chosen such that the exact solution is given by [68]

$$\begin{cases} \Phi = e^t(x^2+y^2)^{0.1}, \\ c_1 = \dfrac{e^t}{(100x)^2+(100y)^2+1}, \\ c_2 = -\dfrac{e^t}{(100x)^2+(100y)^2+1}. \end{cases} \tag{43}$$

Obviously, the exact solutions have a singularity $(0,0)$, which can be visualized clearly in Fig. 2.

Fig. 3 shows the fixed uniform mesh and the redistributed meshes controlled by the monitor functions $\boldsymbol{M}_1, \boldsymbol{M}_2$ at $t = 0.5$ respectively. It is easy to see that the redistributed meshes are capable of capturing the position of the singularity accurately. Furthermore, $\boldsymbol{M}_2$ is able to drive the mesh nodes closer towards the singularity than $\boldsymbol{M}_1$, resulting in higher accuracy as expected.

When we choose a time step $\delta t = O(h)$, where h is the initial uniform mesh size, the analysis for finite element approximation shows that the optimal convergence order for H^1 norm is $O(\text{DOF}^{\frac{1}{2}})$ for linear finite element discretization in two-dimensional problem. Fig. 4 demonstrates the convergence plot in $L^1(0,T;H^1(\Omega))$ norm. Clearly, the redistributed mesh outperforms the fixed uniform mesh in both convergence speed and accuracy, irrespective of the monitor function employed. Additionally, as shown in Fig. 5, the redistributed mesh requires less CPU time than the fixed uniform mesh to achieve comparable accuracy. The results also show that the new monitor function $\boldsymbol{M}_2$ is superior to the traditional monitor function $\boldsymbol{M}_1$ in this case.

Besides, to further illustrate the time cost of the moving mesh method, we provide the CPU times for solving the PNP system and moving mesh in Table 1. We observe that the time cost of the moving mesh is acceptable for smaller-scale grids. However, for larger-scale grids, the time cost significantly increases. Despite this, the considerable improvement in solution accuracy justifies this time cost. In future work, we will further optimize the algorithm to reduce the time expenditure.

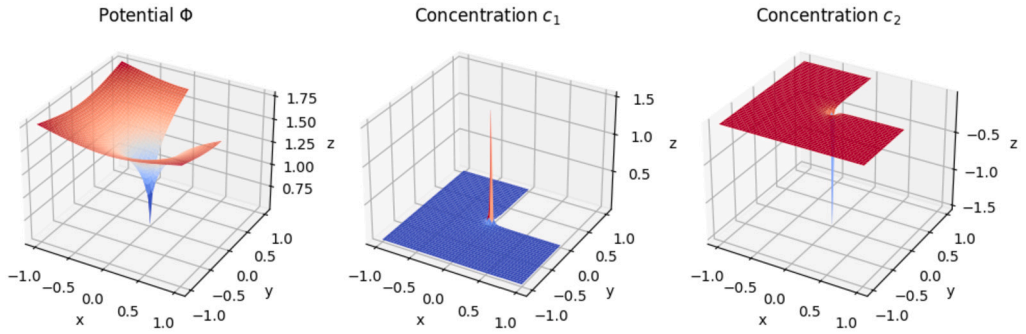
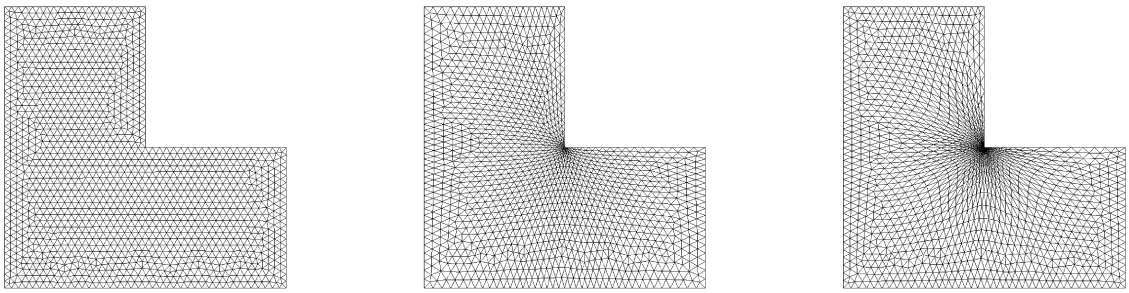


Fig. 2. The exact solution of Φ, c_1 and c_2 when $t = 0.5$ for Example 6.1.



(a) fixed mesh (b) redistributed mesh with $\boldsymbol{M}_1$ (c) redistributed mesh with $\boldsymbol{M}_2$

Fig. 3. Comparison chart of fixed mesh and redistributed meshes when time $t = 0.5$ and element scale $h = 0.05$.

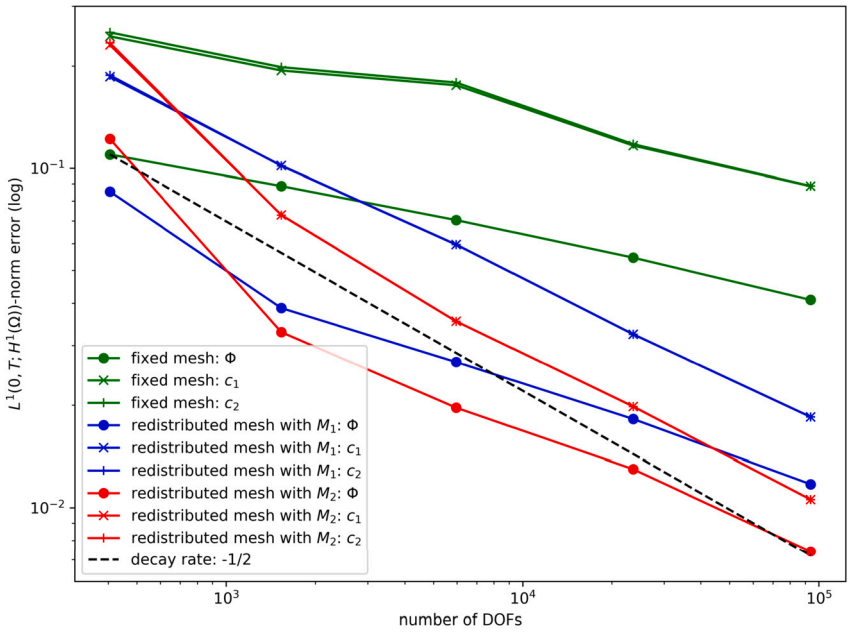


Fig. 4. Convergence plot in $L^1(0,T;H^1(\Omega))$ error.

6.2. Example 2

The second example introduces a case with internal layers. The computational domain for this example is a square region $\Omega = [0,1] \times [0,1]$, and the time span is $T = 0.5$. The basic parameters for the PNP equation are the same as in Example 6.1. Suitable boundary conditions and source terms $f_k, k = 0,1,2$ are chosen to ensure that the exact solution is given by:

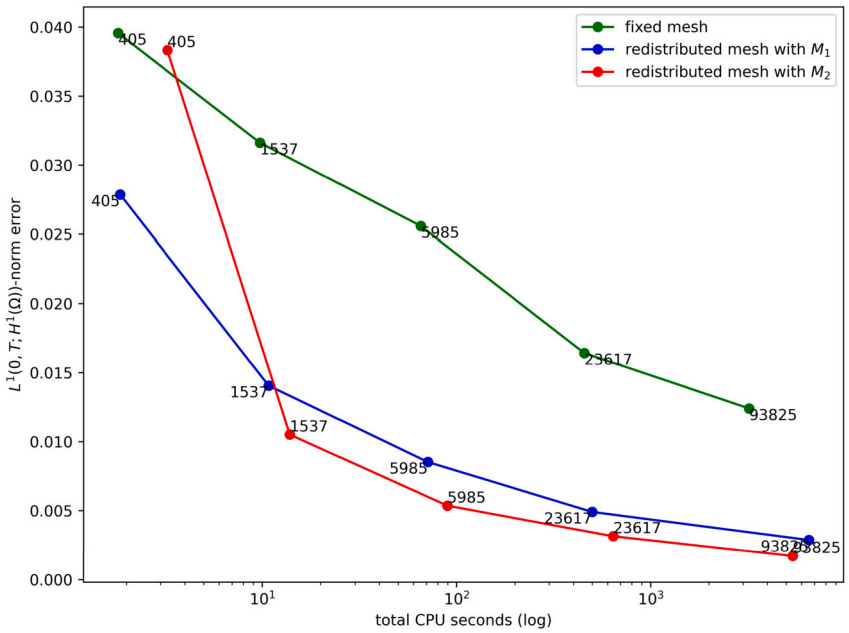


Fig. 5. Relative error-CPU seconds curve for Example 6.1.

Table 1
Three parts of CPU times for Example 6.1.

	NDOF				
	405	1537	5985	23617	93825
CPU seconds for solving PNP system					
fixed mesh	1.8067	9.7406	65.375	455.18	3223.1
redistributed mesh with M_1	1.5719	9.5674	65.067	462.05	3260.9
redistributed mesh with M_2	1.6136	9.2696	64.48	447.39	3230.5
CPU seconds for moving mesh					
fixed mesh	0	0	0	0	0
redistributed mesh with M_1	0.2800	1.2297	5.9607	38.060	3291.1
redistributed mesh with M_2	1.6300	4.6122	24.953	193.54	2172.6
Total CPU seconds					
fixed mesh	1.8067	9.7406	65.375	455.18	3223.1
redistributed mesh with M_1	1.8519	10.797	71.028	500.11	6551.9
redistributed mesh with M_2	3.2436	13.882	89.433	640.93	5403.1

$$\begin{cases} \Phi = \exp\left(\dfrac{(x-0.25)^2}{0.001e^{-t}}\right) + \exp\left(\dfrac{(y-0.25)^2}{0.001e^{-t}}\right), \\ c_1 = 10\exp\left(\dfrac{(x-0.25)^2}{0.0005e^{-t}}\right), \\ c_2 = -10\exp\left(\dfrac{(y-0.25)^2}{0.0005e^{-t}}\right). \end{cases} \tag{44}$$

In this example, the internal layer is controlled by the exponential term e^{-t} of the Gaussian functions. When t increases, the boundary layer gradually thins. When $t = 0.5$, the exact solution is shown in Fig. 6.

Although the solutions are globally smooth in this example, the presence of the internal layer causes the finite element method with fixed mesh to achieve the expected convergence rate but with large convergence constants. However, the moving mesh method is a useful choice to improve the efficiency of convergence, reducing the CPU time required to achieve the same accuracy, as shown in Fig. 7. From the figure, we can also observe that although both Monitor functions achieve similar accuracy with the same degrees of freedom, $\boldsymbol{M}_2$ consumes less CPU time than $\boldsymbol{M}_1$.

Additionally, it is worth noting that the exponential composition in the construction of the analytic solution is natural. In practical applications of the Poisson–Nernst–Planck (PNP) equation, many potential wells formed by charges exhibit exponential forms. For potential wells with more exponential term combinations, we can naturally extend them as variations of Example 6.2, where the effectiveness of the moving mesh method remains applicable.

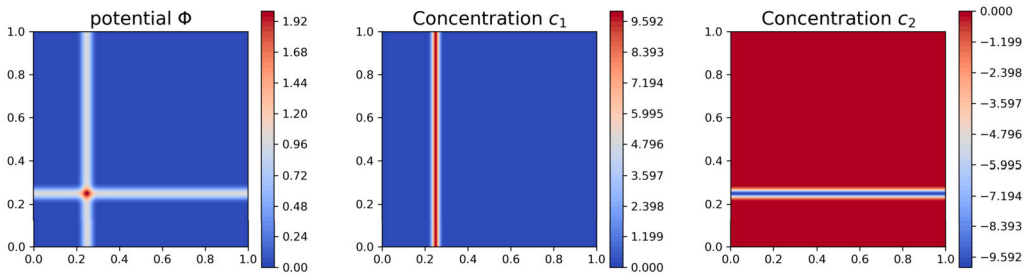


Fig. 6. Exact solutions Φ, c_1, and c_2 at $t = 0.5$ for Example 6.2.

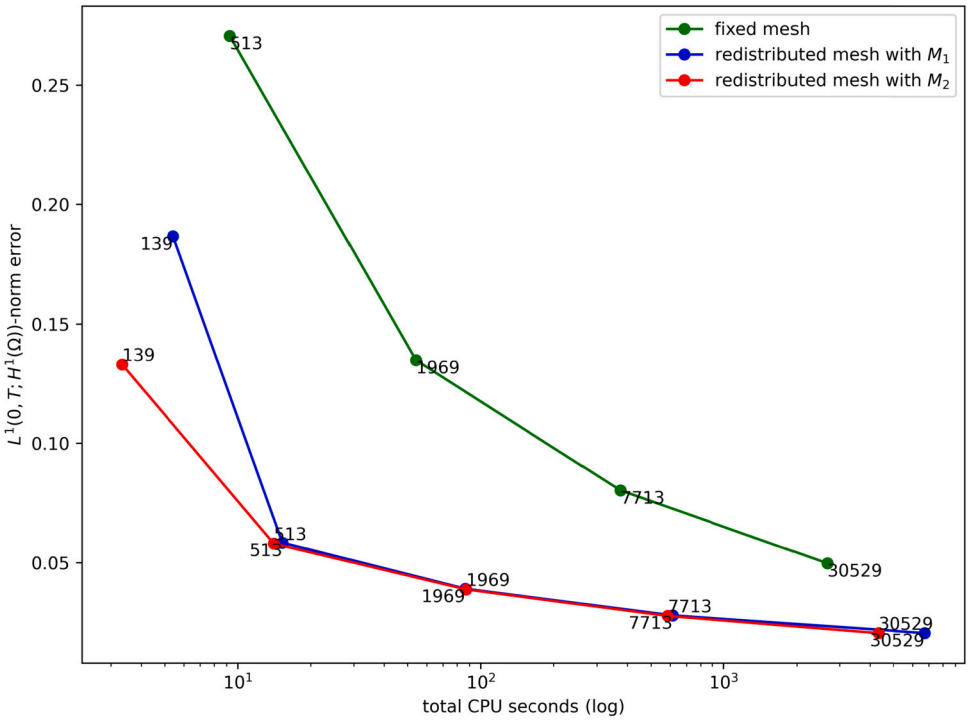


Fig. 7. Relative error-CPU seconds curve for Example 6.2.

6.3. Example 3

The third case involves a PNP system that features a moving singularity. This system can be mathematically represented as follows

$$\begin{cases} \dfrac{\partial c_k}{\partial t} - \nabla \cdot (D_k(\nabla c_k + z_k c_k \nabla \Phi)) = f_k, \quad k = 1,2, \text{ in } \Omega, t \in [0,T], \\ -\nabla \cdot (\epsilon \nabla \Phi) = z_1 c_1 + z_2 c_2 + f_0, \qquad\qquad \text{in } \Omega, t \in [0,T], \end{cases} \tag{45}$$

where the computational region $\Omega = [-1,1]^2 \backslash (-1.0 \times 10^{-4}, 1] \times (-1.0 \times 10^{-4}, 1] \in \mathbb{R}^2$ and the time span $T = 0.5$. The basic parameters for PNP equations are the same as in Example 6.1. Select appropriate boundary conditions and source terms $f_k, k = 0,1,2$, such that the exact solution is represented as [68]

$$\begin{cases} \Phi = (x^2 + (y - 0.4t)^2)^{0.1}, \\ c_1 = \ln \Phi, \\ c_2 = -\ln \Phi. \end{cases} \tag{46}$$

The exact solution (46) reveals that the position of the singularity is $(0, 0.4t)$, which moves with time t. When $t = 0.5$, it can be illustrated in Fig. 8.

According to Fig. 9, it can be observed that the moving mesh method exhibits significantly improved convergence rates compared to the fixed mesh. Particularly, when using $\boldsymbol{M}_2$ as the monitor function, the convergence rate reaches the theoretical optimum. In

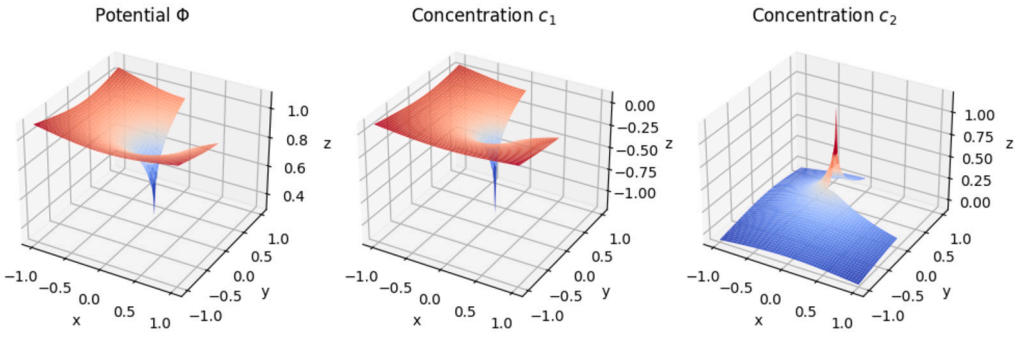


Fig. 8. The exact solution of Φ, c_1 and c_2 when $t = 0.5$ for Example 6.3.

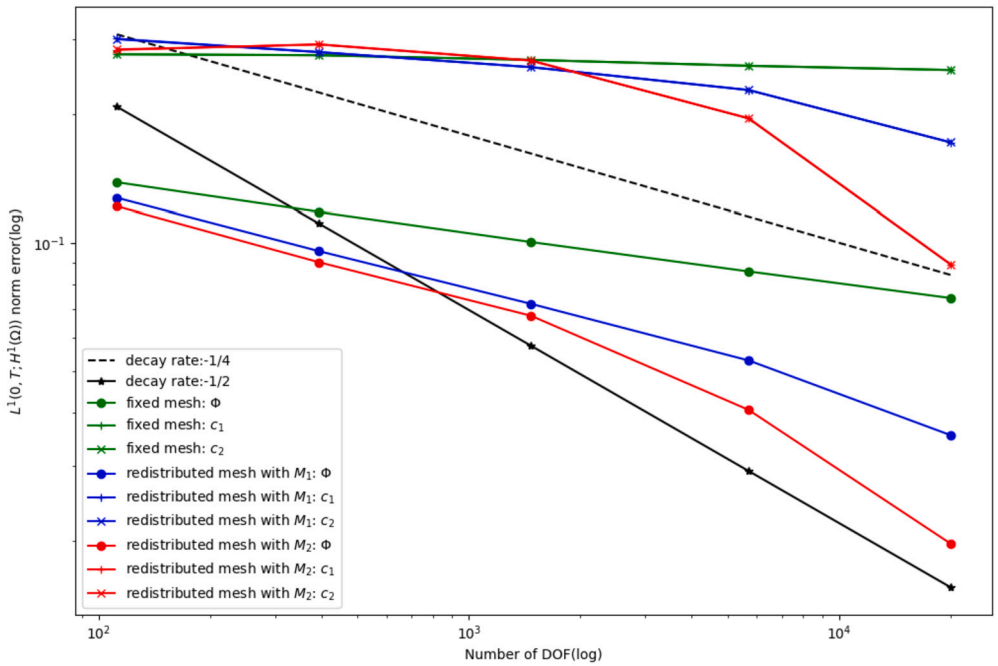


Fig. 9. $L^1(0,T;H^1(\Omega))$ error and convergence rate. (The curves of c_1 and c_2 are overlapped.)

contrast to Example 6.1, this example better demonstrates the capability of moving mesh methods to capture the dynamic singularity. Furthermore, it showcases the effectiveness and superiority of the newly designed monitor function $\boldsymbol{M}_2$.

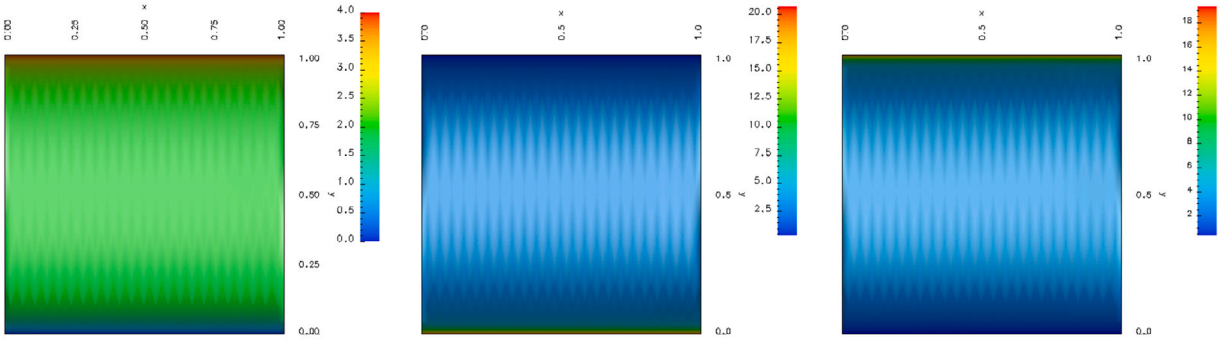
6.4. Example 4

In this example, we consider the standard PNP equations which have a convection-dominant effect at the boundary with a Dirichlet boundary condition,

$$\begin{cases} \dfrac{\partial c_k}{\partial t} = \nabla \cdot (D_k(\nabla c_k + z_k c_k \nabla \Phi)) \quad k = 1,2, \ \text{in } \Omega, t \in [0,T], \\ -\nabla \cdot (\epsilon \nabla \Phi) = z_1 c_1 + z_2 c_2 \qquad\qquad\qquad \text{in } \Omega, t \in [0,T], \end{cases} \tag{47}$$

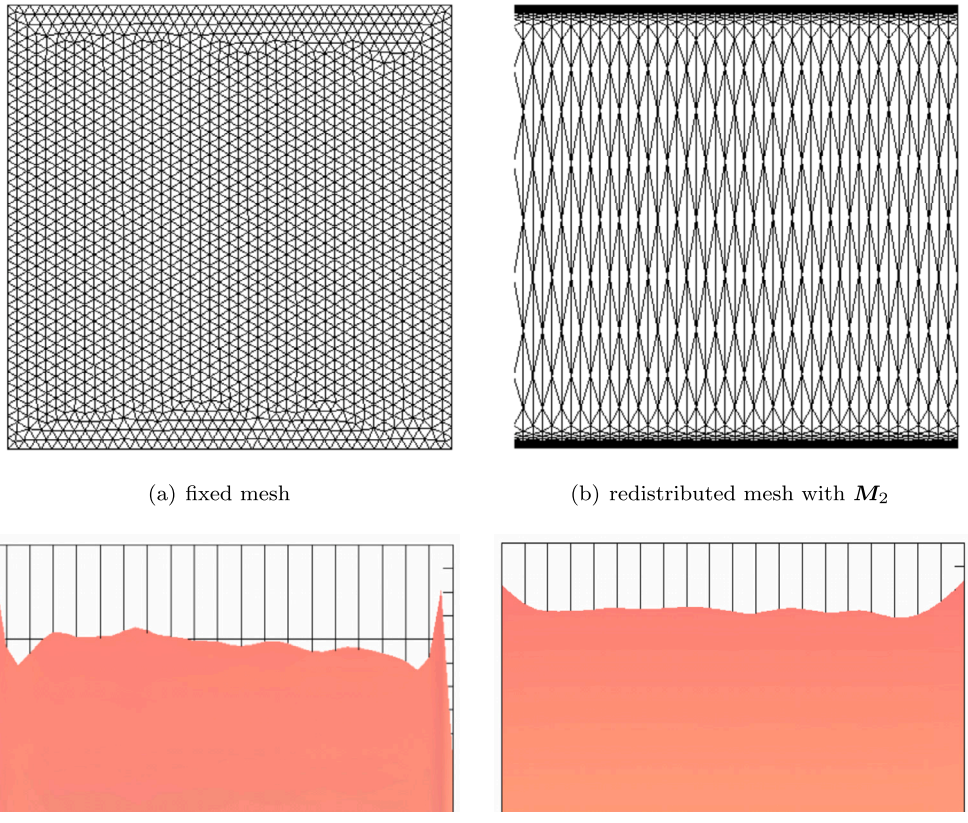
where $\Omega = [0,1]^2 \in \mathbb{R}^2$, $T = 0.5$, $\epsilon = 0.01$, $D_1 = D_2 = 1.0$, $z_1 = 1$, and $z_2 = -1$. As for the initial and boundary conditions, they are defined as follows

$$\begin{cases} c_1(x,y,0) = 3.5, \\ c_2(x,y,0) = 3.5, \end{cases} \tag{48}$$



(a) Φ (b) c_1 (c) c_2

Fig. 10. Illustration of the numerical solutions in equilibrium state solved by moving mesh method with $\boldsymbol{M}_2$.



(a) fixed mesh (b) redistributed mesh with $\boldsymbol{M}_2$

(c) Zoomed-in view of c_2 at Γ_3 for the fixed mesh (d) Zoomed-in view of c_2 at Γ_3 for the redistributed mesh with $\boldsymbol{M}_2$

Fig. 11. Comparison chart of fixed mesh and redistributed meshes when time $t = 1$ and initial element scale $h = 0.025$.

$$\begin{cases} \Phi = 0 & (x,y) \in \Gamma_1, t \in [0,T], \\ \Phi = 4 & (x,y) \in \Gamma_3, t \in [0,T], \\ \epsilon \frac{\partial \Phi}{\partial \boldsymbol{n}} = 0 & (x,y) \in \Gamma_2 \cup \Gamma_4, t \in [0,T], \\ \boldsymbol{J_1} \cdot \boldsymbol{n} = 0 & (x,y) \in \partial\Omega, t \in [0,T], \\ \boldsymbol{J_2} \cdot \boldsymbol{n} = 0 & (x,y) \in \partial\Omega, t \in [0,T], \end{cases} \tag{49}$$

where $\Gamma_1 = \{(x,y) \in \partial\Omega | y = 0\}, \Gamma_2 = \{(x,y) \in \partial\Omega | x = 1\}, \Gamma_3 = \{(x,y) \in \partial\Omega | y = 1\}, \Gamma_4 = \{(x,y) \in \partial\Omega | x = 0\}$.

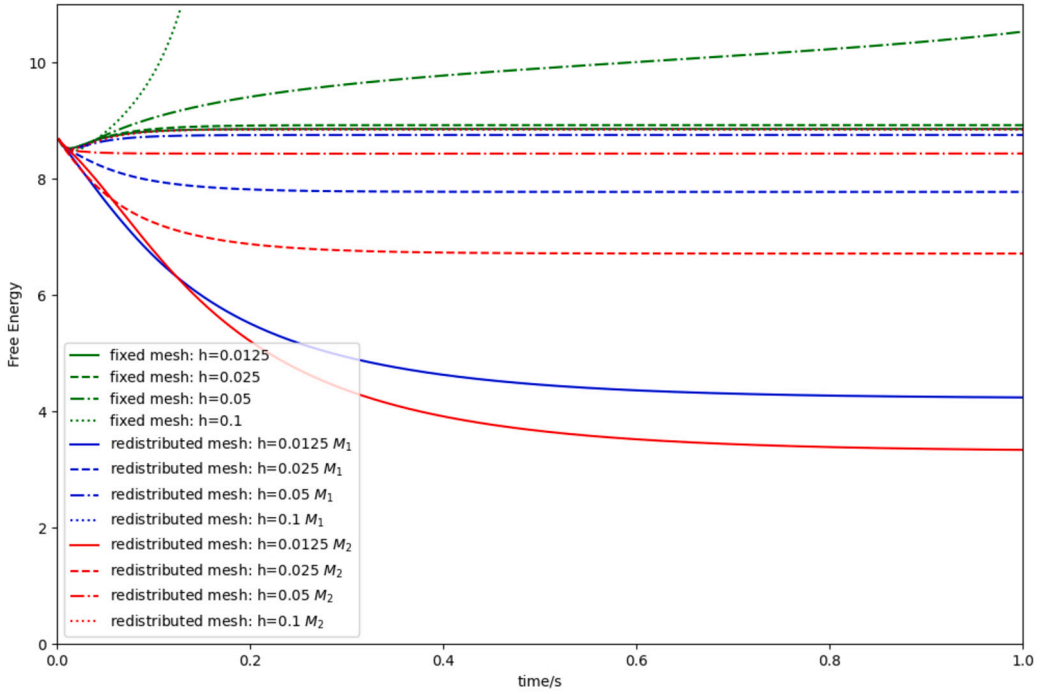


Fig. 12. The free energy for different h when time step $\delta t = 0.002$.

As we can see from Fig. 10, the system displays clear boundary layers around Γ_1 and Γ_3, where the convective term with the velocity vector $\nabla\Phi$ is significantly large within narrow regions. To capture these boundary layers and improve solution accuracy, the moving mesh method is employed. Fig. 11 shows zoomed-in views of the solution c_2 varying with x at the boundary Γ_3 for fixed and redistributed meshes. Here, the figure only shows the $\boldsymbol{M}_2$ result as the visualization of $\boldsymbol{M}_1$ is similar. The figure clearly displays the effectiveness of the mesh redistribution.

In this example, we fix the time step size as $\delta t = 0.002$. Taking free energy as an evaluation metric, the results are shown in Fig. 12. It can be observed that in the case of sparse mesh, if the mesh is fixed, unbounded energy occurs during time marching, while the moving mesh method can effectively redistribute mesh points and keep the energy bounded. In the case of a denser mesh, the moving mesh method meets the energy dissipation property, where the free energy monotonically decreases over time. However, the fixed mesh fails to capture the energy dissipation property even with a denser mesh (initial uniform mesh size $h = 0.0125$). Additionally, based on the monitor function $\boldsymbol{M}_2$, the mesh redistributed by the moving mesh method yields a smaller equilibrium energy compared to $\boldsymbol{M}_1$. This indicates that the finite element space corresponding to the mesh redistribution based on $\boldsymbol{M}_2$ provides a better approximation for the solutions to some degree.

7. Conclusion

In this paper, we apply a moving mesh method to address the challenges posed in the numerical solution of the Poisson–Nernst–Planck (PNP) equations. The PNP equations are decoupled using the Gummel iteration technique. Spatial discretization was achieved through a finite element approach, while temporal discretization was realized using an implicit scheme. We presented a comprehensive derivation and explanation of the moving mesh method, which includes a relatively general functional and a novel flux-based monitor function. Our experiments display clearly that the redistributed mesh generated by moving mesh method is superior to the fixed mesh in solving PNP equations with singularities or exhibiting boundary layers. Additionally, the flux-based monitor function we proposed outperforms the traditional counterpart in all the numerical examples of this paper.

In future work, we aim to combine the moving mesh method with more stable numerical algorithms. Furthermore, we will employ the moving mesh technique to other systems of transport problem where flux exists.

CRediT authorship contribution statement

Minrui Lv: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Benzhuo Lu:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization, Methodology.

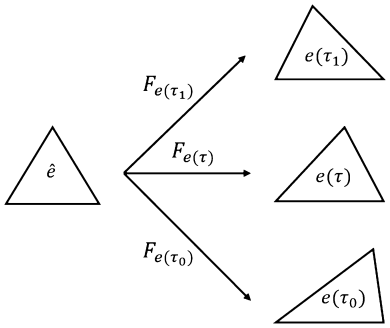


Fig. A.13. Illustration of $F_{e(\tau)} : \hat{e} \to e(\tau)$.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Minrui Lv and Benzhuo Lu reports financial support was provided by National Natural Science Foundation of China.

Data availability

Data will be made available on request.

Acknowledgements

This research was funded by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0500000) and the National Natural Science Foundation of China (Grant No. 12371413, No. 22073110). We would like to thank the students Yan Xie, Sheng Gui for their valuable comments and suggestions.

Appendix A. The derivation of (30)

Set $\mathcal{T}(\tau) = \{\mathcal{V}_{\mathcal{T}(\tau)} = \{\boldsymbol{x}_i(\tau)\}_{i=1}^{N_v}, \mathcal{E}_{\mathcal{T}(\tau)} = \{e_i(\tau)\}_{i=1}^{N_e}\}$ be the mesh at τ during the process of mesh moving. Due to the topology of mesh being invariant, any mesh element $e(\tau) \in E_{\mathcal{T}h(\tau)}$ can be regarded as the image of a fixed reference element $\hat{e}$ under a family of invertible affine mappings $F_{e(\tau)}$, i.e.,

$$\begin{aligned} &F_{e(\tau)} : \hat{e} \to e(\tau), \\ &\qquad \boldsymbol{\xi} \mapsto \boldsymbol{x} = F_{e(\tau)}(\boldsymbol{\xi}), \end{aligned} \tag{A.1}$$

which in the two-dimensional case is clearly depicted in Fig. A.13. We can observe that the relationship between any basis function φ_j restricted to $e(\tau)$ and its corresponding basis $\hat{\varphi}_j$ restricted to $\hat{e}$ is as follows,

$$\varphi_j(\boldsymbol{x}(\tau), \tau) = \hat{\varphi}_j \circ F_{e(\tau)}^{-1}(\boldsymbol{x}) = \hat{\varphi}_j(\boldsymbol{\xi}). \tag{A.2}$$

It means that the material derivative of any basis in $V_{\mathcal{T}}^k(\Omega)$ is vanished, i.e.,

$$\frac{\mathrm{d}\varphi_j(\boldsymbol{x}(\tau), \tau)}{\mathrm{d}\tau} = \frac{\partial \hat{\varphi}_j(\boldsymbol{\xi})}{\partial \tau} = 0, \qquad j = 1, 2, \ldots, N. \tag{A.3}$$

According to (A.3) and the relationship between material and local derivative, we can derive (30) through the following steps, i.e.,

$$\begin{aligned} 0 = \frac{\mathrm{d}\varphi_j}{\mathrm{d}\tau} &= \frac{\partial \varphi_j}{\partial \tau} + \nabla\varphi_j \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\tau} \\ &= \frac{\partial \varphi_j}{\partial \tau} + \nabla\varphi_j \cdot \frac{\mathrm{d}}{\mathrm{d}\tau}\Big(\sum_{k=1}^{N_v} \phi_k \boldsymbol{x}_k(\tau)\Big) && \text{(linear basis function)} \\ &= \frac{\partial \varphi_j}{\partial \tau} + \nabla\varphi_j \cdot \sum_{k=1}^{N_v} \Big(\frac{\mathrm{d}\phi_k}{\mathrm{d}\tau}\boldsymbol{x}_k(\tau) + \phi_k \frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\tau}\Big) \\ &= \frac{\partial \varphi_j}{\partial \tau} + \nabla\varphi_j \cdot \sum_{k=1}^{N_v} \phi_k \frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\tau} && \Big(\frac{\mathrm{d}\phi_k}{\mathrm{d}\tau} = \frac{\partial \phi_k}{\partial \tau}\Big|_{\xi} = 0\Big) \\ &= \frac{\partial \varphi_j}{\partial \tau} + \nabla\varphi_j \cdot \delta\boldsymbol{x}, \end{aligned} \tag{A.4}$$

where $\delta\boldsymbol{x}, \phi_k$ have the same definitions as those in (30).

References

[1] F. Alauzet, A. Loseille, A decade of progress on anisotropic mesh adaptation for computational fluid dynamics, Comput. Aided Des. 72 (2016) 13–39.
[2] R.E. Bank, J.F. Bürgler, W. Fichtner, R.K. Smith, Some upwinding techniques for finite element approximations of convection–diffusion equations, Numer. Math. 58 (1990) 185–202.
[3] M. Bessemoulin-Chatard, C. Chainais-Hillairet, M.H. Vignal, Study of a finite volume scheme for the drift-diffusion system. Asymptotic behavior in the quasi-neutral limit, SIAM J. Numer. Anal. 52 (2014) 1666–1691.
[4] P. Bochev, M. Perego, K. Peterson, Formulation and analysis of a parameter–free stabilized finite element method, SIAM J. Numer. Anal. 53 (2015) 2363–2388.
[5] D. Boffi, N. Cavallini, F. Gardini, L. Gastaldi, Local mass conservation of Stokes finite elements, J. Sci. Comput. 52 (2012) 383–400.
[6] J. Brackbill, An adaptive grid with directional control, J. Comput. Phys. 108 (1993) 38–50.
[7] F. Brezzi, L. Marini, S. Micheletti, P. Pietra, R. Sacco, S. Wang, Discretization of semiconductor device problems (I), Handb. Numer. Anal. 13 (2005) 317–441.
[8] F. Brezzi, L.D. Marini, P. Pietra, Two-dimensional exponential fitting and applications to drift–diffusion models, SIAM J. Numer. Anal. 26 (1989) 1342–1355.
[9] A.N. Brooks, T.J. Hughes, Streamline upwind/Petrov–Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations, Comput. Methods Appl. Mech. Eng. 32 (1982) 199–259.
[10] W. Cao, W. Huang, R.D. Russell, A study of monitor functions for two–dimensional adaptive mesh generation, SIAM J. Sci. Comput. 20 (1999) 1978–1994.
[11] C. Chainais-Hillairet, Y.J. Peng, Finite volume approximation for degenerate drift–diffusion system in several space dimensions, Math. Models Methods Appl. Sci. 14 (2004) 461–481.
[12] L. Chen, P. Sun, J. Xu, Optimal anisotropic meshes for minimizing interpolation errors in L^p–norm, Math. Comput. 76 (2007) 179–204.
[13] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, Society for Industrial and Applied Mathematics, 2002.
[14] F. Ciucci, W. Lai, Derivation of micro/macro lithium battery models from homogenization, Transp. Porous Media 88 (2011) 249–270.
[15] A. van Dam, P.A. Zegeling, et al., Balanced monitoring of flow phenomena in moving mesh methods, Commun. Comput. Phys. 7 (2010) 138.
[16] P. Das, Comparison of a priori and a posteriori meshes for singularly perturbed nonlinear parameterized problems, Am. J. Comput. Appl. Math. 290 (2015) 16–25.
[17] P. Das, S. Natesan, Adaptive mesh generation for singularly perturbed fourth-order ordinary differential equations, Int. J. Comput. Math. 92 (2015) 562–578.
[18] P. Das, J. Vigo-Aguiar, Parameter uniform optimal order numerical approximation of a class of singularly perturbed system of reaction diffusion problems involving a small perturbation parameter, Am. J. Comput. Appl. Math. 354 (2019) 533–544.
[19] M. Delfour, G. Payre, J.P. Zolésio, An optimal triangulation for second–order elliptic problems, Comput. Methods Appl. Mech. Eng. 50 (1985) 231–261.
[20] Y. Di, R. Li, T. Tang, P. Zhang, Moving mesh finite element methods for the incompressible Navier–Stokes equations, SIAM J. Sci. Comput. 26 (2005) 1036–1056.
[21] Y. Di, R. Li, T. Tang, et al., A general moving mesh framework in 3D and its application for simulating the mixture of multi–phase flows, Commun. Comput. Phys. 3 (2008) 582–602.
[22] J. Ding, Z. Wang, S. Zhou, Positivity preserving finite difference methods for Poisson–Nernst–Planck equations with steric interactions: application to slit–shaped nanopore conductance, J. Comput. Phys. 397 (2019) 108864.
[23] C. Dobrzynski, P. Frey, Anisotropic Delaunay mesh adaptation for unsteady simulations, in: Proceedings of the 17th International Meshing Roundtable, Springer, 2008, pp. 177–194.
[24] A.S. Dvinsky, Adaptive grid generation from harmonic maps on Riemannian manifolds, J. Comput. Phys. 95 (1991) 450–476.
[25] R. Eymard, T. Gallouët, R. Herbin, Finite volume methods, Handb. Numer. Anal. 7 (2000) 713–1018.
[26] A. Flavell, M. Machen, B. Eisenberg, J. Kabre, C. Liu, X. Li, A conservative finite difference scheme for Poisson–Nernst–Planck equations, J. Comput. Electron. 13 (2014) 235–249.
[27] H. Gao, P. Sun, A linearized local conservative mixed finite element method for Poisson–Nernst–Planck equations, J. Sci. Comput. 77 (2018) 793–817.
[28] J.L. Gracia, E. O'Riordan, Numerical approximations to a singularly perturbed convection–diffusion problem with a discontinuous initial condition, Numer. Algorithms 88 (2021) 1851–1873.
[29] D. He, K. Pan, An energy preserving finite difference scheme for the Poisson–Nernst–Planck system, Appl. Math. Comput. 287 (2016) 214–223.
[30] J.J. Heys, E. Lee, T.A. Manteuffel, S.F. McCormick, An alternative least–squares formulation of the Navier–Stokes equations with improved mass conservation, J. Comput. Phys. 226 (2007) 994–1006.
[31] U. Hollerbach, D.P. Chen, R.S. Eisenberg, Two- and three-dimensional Poisson–Nernst–Planck simulations of current flow through gramicidin A, J. Sci. Comput. 16 (2001) 373–409.
[32] W. Huang, L. Kamenski, J. Lang, A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates, J. Comput. Phys. 229 (2010) 2179–2198.
[33] W. Huang, R.D. Russell, A high dimensional moving mesh strategy, Appl. Numer. Math. 26 (1998) 63–76.
[34] W. Huang, R.D. Russell, Moving mesh strategy based on a gradient flow equation for two–dimensional problems, SIAM J. Sci. Comput. 20 (1998) 998–1015.
[35] W. Huang, R.D. Russell, Adaptive Moving Mesh Methods, vol. 174, Springer Science & Business Media, 2010.
[36] D. Kim, H. Choi, A second-order time-accurate finite volume method for unsteady incompressible flow on hybrid unstructured grids, J. Comput. Phys. 162 (2000) 411–428.
[37] Y. Kuang, G. Hu, An adaptive FEM with ITP approach for steady Schrödinger equation, Int. J. Comput. Math. 95 (2018) 187–201.
[38] M.G. Kurnikova, R.D. Coalson, P. Graf, A. Nitzan, A lattice relaxation algorithm for three-dimensional Poisson–Nernst–Planck theory with application to ion transport through the gramicidin A channel, Biophys. J. 76 (1999) 642–656.
[39] R. Li, W. Liu, H. Ma, T. Tang, Adaptive finite element approximation for distributed elliptic optimal control problems, SIAM J. Control Optim. 41 (2002) 1321–1349.
[40] R. Li, T. Tang, P. Zhang, A moving mesh finite element algorithm for singular problems in two and three space dimensions, J. Comput. Phys. 177 (2002) 365–393.
[41] C. Liu, C. Wang, S.M. Wise, X. Yue, S. Zhou, A positivity–preserving, energy stable and convergent numerical scheme for the Poisson–Nernst–Planck system, Math. Comput. 90 (2021) 2071–2106.
[42] H. Liu, Z. Wang, A free energy satisfying finite difference method for Poisson–Nernst–Planck equations, J. Comput. Phys. 268 (2014) 363–376.
[43] L. Liu, Y. Chen, A posteriori error estimation in maximum norm for a strongly coupled system of two singularly perturbed convection–diffusion problems, Am. J. Comput. Appl. Math. 313 (2017) 152–167.
[44] X. Liu, Y. Qiao, B. Lu, Analysis of the mean field free energy functional of electrolyte solution with nonhomogenous boundary conditions and the generalized PB/PNP equations with inhomogeneous dielectric permittivity, SIAM J. Appl. Math. 78 (2018) 1131–1154.
[45] B. Lu, M.J. Holst, J.A. McCammon, Y. Zhou, Poisson–Nernst–Planck equations for simulating biomolecular diffusion–reaction processes I: finite element solutions, J. Comput. Phys. 229 (2010) 6979–6994.
[46] G. MacDonald, J. Mackenzie, M. Nolan, R. Insall, A computational method for the coupled solution of reaction–diffusion equations on evolving domains and manifolds: application to a model of cell migration and Chemotaxis, J. Comput. Phys. 309 (2016) 207–226.
[47] P.A. Markowich, M.A. Zlámal, Inverse–average–type finite element discretizations of selfadjoint second-order elliptic problems, Math. Comput. 51 (1988) 431–449.
[48] S.R. Mathur, J.Y. Murthy, A multigrid method for the Poisson–Nernst–Planck equations, Int. J. Heat Mass Transf. 52 (2009) 4031–4039.

[49] J. Miller, W. Schilders, S. Wang, Application of finite element methods to the simulation of semiconductor devices, Rep. Prog. Phys. 62 (1999) 277.
[50] K. Miller, R.N. Miller, Moving finite elements. I, SIAM J. Numer. Anal. 18 (1981) 1019–1032.
[51] Y. Qiu, D. Sloan, T. Tang, Numerical solution of a singularly perturbed two–point boundary value problem using equidistribution: analysis of convergence, Am. J. Comput. Appl. Math. 116 (2000) 121–143.
[52] G. Richardson, J. King, Time-dependent modelling and asymptotic analysis of electrochemical cells, J. Eng. Math. 59 (2007) 239–275.
[53] P. Thomas, C. Lombard, Geometric conservation law and its application to flow computations on moving grids, AIAA J. 17 (1979) 1030–1037.
[54] Y. Tourigny, F. Hülsemann, A new moving mesh algorithm for the finite element solution of variational problems, SIAM J. Numer. Anal. 35 (1998) 1416–1438.
[55] B. Tu, M. Chen, Y. Xie, L. Zhang, B. Eisenberg, B. Lu, A parallel finite element simulator for ion transport through three–dimensional ion channel systems, J. Comput. Chem. 34 (2013) 2065–2078.
[56] D. Wang, X.P. Wang, A three-dimensional adaptive method based on the iterative grid redistribution, J. Comput. Phys. 199 (2004) 423–436.
[57] Q. Wang, H. Li, L. Zhang, B. Lu, A stabilized finite element method for the Poisson–Nernst–Planck equations in three-dimensional ion channel simulations, Appl. Math. Lett. 111 (2021) 106652.
[58] A.M. Winslow, Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh, J. Comput. Phys. 1 (1966) 149–172.
[59] A.M. Winslow, Adaptive–mesh zoning by the equipotential method, Technical Report, Lawrence Livermore National Lab., CA (USA), 1981.
[60] S. Wu, J. Xu, Simplex–averaged finite element methods for H(grad), H(curl), and H(div) convection–diffusion problems, SIAM J. Numer. Anal. 58 (2020) 884–906.
[61] D. Xie, Z. Chao, A Poisson–Nernst–Planck single ion channel model and its effective finite element solver, J. Comput. Phys. 481 (2023) 112043.
[62] D. Xie, B. Lu, An effective finite element iterative solver for a Poisson–Nernst–Planck ion channel model with periodic boundary conditions, SIAM J. Sci. Comput. 42 (2020) B1490–B1516.
[63] Y. Xie, T. Liu, B. Tu, B. Lu, L. Zhang, Automated parallel and body–fitted mesh generation in finite element simulation of macromolecular systems, Commun. Comput. Phys. 19 (2016) 582–602.
[64] G. Xu, B. Li, L. Shu, L. Chen, J. Xu, T. Khajah, Efficient R-adaptive isogeometric analysis with Winslow's mapping and monitor function approach, Am. J. Comput. Appl. Math. 351 (2019) 186–197.
[65] J. Xu, L. Zikatanov, A monotone finite element scheme for convection–diffusion equations, Math. Comput. 68 (1999) 1429–1446.
[66] J. Ying, R. Fan, J. Li, B. Lu, A new block preconditioner and improved finite element solver of Poisson–Nernst–Planck equation, J. Comput. Phys. 430 (2021) 110098.
[67] Q. Zhang, Q. Wang, L. Zhang, B. Lu, A class of finite element methods with averaging techniques for solving the three-dimensional drift–diffusion model in semiconductor device simulations, J. Comput. Phys. 458 (2022) 111086.
[68] W. Zhu, Y. Yang, G. Ji, B. Lu, Residual type a posteriori error estimates for the time-dependent Poisson–Nernst–Planck equations, J. Sci. Comput. 90 (2022) 1–35.