Advances in Applied Mathematics and Mechanics
Adv. Appl. Math. Mech., Vol. **4**, No. 2, pp. 156-174

DOI: 10.4208/aamm.11-m1192
April 2012

The Eulerian-Lagrangian Method with Accurate Numerical Integration

Kun Li*

LMAM & School of Mathematical Sciences, Peking University, Beijing 100080, China

Received 8 May 2011; Accepted (in revised version) 21 June 2011

Available online 26 March 2012

Abstract. This paper is devoted to the study of the Eulerian-Lagrangian method (ELM) for convection-diffusion equations on unstructured grids with or without accurate numerical integration. We first propose an efficient and accurate algorithm to calculate the integrals in the Eulerian-Lagrangian method. Our approach is based on an algorithm for finding the intersection of two non-matching grids. It has optimal algorithmic complexity and runs fast enough to make time-dependent velocity fields feasible. The evaluation of the integrals leads to increased precision and the unconditional stability. We demonstrate by numerical examples that the ELM with our proposed algorithm for accurate numerical integration has the following two features: first it is much more accurate and more stable than the ones with traditional numerical integration techniques and secondly the overall cost of the proposed method is comparable with the traditional ones.

AMS subject classifications: 65D18, 65D30, 65M25, 65N30

Key words: Eulerian-Lagrangian method, intersection of non-matching grids, exact integration.

1 Introduction

The Eulerian-Lagrangian Method (ELM), also called Semi-Lagrangian Method (SLM), is known as an efficient numerical method for both pure convection and convection-diffusion problems. It works by interpreting the convection term and the time derivative as a material derivative. An implicit discretization based on this approach results in an unconditionally stable scheme that leads to a symmetric positive definite system. The price is a more complicated implementation. In particular, it is noted that inner products of test functions with the solution of the previous time step propagated along

*Corresponding author.
URL: http://dsec.pku.edu.cn/∼kli/index_ch.htm
Email: kli@math.pku.edu.cn (K. Li)

the integral lines of the velocity field. However, the inner products generally can not be evaluated exactly.

The convergence analysis for the Eulerian-Lagrangian method has been established in [8] for the pure convection problem, and in [2,9] for the convection-diffusion problem. In these works it is assumed that the underlying integrals are evaluated exactly. As demonstrated in [7], ELM may not converge if numerical integration is not accurate enough. Indeed, both theoretical and numerical results show that the numerical approximation of the integration may lead to numerical instability, unless quadrature rules of sufficiently high order are used. Finite element interpolation has been often used for numerical integration, but this approach is known to introduce too much numerical diffusion [5,6,11].

In the implementation of ELM method, inner products that involve two finite element functions on two different grids need to be evaluated. How to accurately evaluate these integration accurately is the main topic of this paper. For two dimensional rectangular grids, an approach called "area weighting" was introduced in [7]. For two dimensional triangular grids, a special algorithm for "exact projection" was introduced in [10]. Attaining the information needed by integration is still a time-consuming task and the algorithm to find the intersection of two non-matching grids is output-sensitive or intersection-sensitive (see [1]).

In this paper we introduce a more general and more efficient algorithm to evaluate the inner products. We followed and improved an algorithm proposed by [4] (which was designed for mortar finite element computations) to find the intersection of two non-matching grids. This type of algorithms work for conforming grids of any space dimensions and for all element types and they appear to be more general and more effective than the earlier methods proposed in the literature (e.g., [3] designed for the moving mesh method).

Using the accurate integration method mentioned above, we will use numerical examples to show the resulting ELM can achieve the optimal convergence rate and we will also demonstrate the accurate integration will improve the precision of the numerical solution with only marginal extra computational complexity and such an improvement is more significant for solution with singularities.

The rest of this paper is organized as follows. In Section 2, we introduce ELM for convection-diffusion equations. In Section 3, we describe a specific algorithm of accurate integration with details of the algorithm for finding the intersection of two non-matching grids in arbitrary dimensional case. We then show how to improve this algorithm. We will also report a number of numerical tests to validate our algorithm in Section 4. In addition, we will also show the importance of the exact integration for the solution space with weak regularity.

2 The Eulerian-Lagrangian method

Let Ω be a domain in $\mathbb{R}^d$ and let $T > 0$. For a function $u : \Omega \times [0, T] \to \mathbb{R}$ we consider

the linear model convection-diffusion equation

$$\frac{\partial u}{\partial t} + b \cdot \nabla u - \epsilon \Delta u = f, \quad \text{on } \Omega \times [0,T], \tag{2.1}$$

with initial condition $u(x,0) = u_0(x)$ on Ω and suitable boundary conditions. The vector field $b : \Omega \times [0,T] \to \mathbb{R}^d$ is called the velocity field. For simplicity we assume that it is divergence-free

$$\nabla \cdot b = 0. \tag{2.2}$$

The parameter ϵ is nonnegative. If it is zero, (2.1) is called a pure convection equation.

By the assumption, the velocity field b is integrable, i.e., there exists a flow mapping

$$y(\cdot, s, t) : \Omega \to \Omega,$$

such that

$$\frac{dy}{ds} = b(y(x,s,t),s), \qquad \forall x \in \Omega, \ \ s,t \in (0,T), \tag{2.3a}$$

$$y(x,t,t) = x, \qquad \forall x \in \Omega, \ \ t \in (0,T). \tag{2.3b}$$

The Eulerian-Lagrangian method treats (2.1) by interpreting the term $\partial u/\partial t + (b \cdot \nabla)u$ as the material derivative Du/Dt,

$$\frac{Du}{Dt} := \lim_{k \to 0} \frac{u(x,t) - u\big(y(x,t-k,t),t-k\big)}{k} = \frac{\partial u}{\partial t} + b \cdot \nabla u.$$

Using the material derivative, (2.1) becomes

$$\frac{Du}{Dt} - \epsilon \Delta u = f. \tag{2.4}$$

Among the many possible implicit schemes for time discretization, we take the backward Euler scheme as an example. For simplicity we assume a constant time step size τ. Call u^n and f^n the discrete solution and right-hand side at time step n. We obtain the time-stepping procedure

$$\frac{u^{n+1} - u^n \circ y(x, t^n, t^{n+1})}{\tau} - \epsilon \Delta u^{n+1} = f^n. \tag{2.5}$$

Let V be a suitable space of test functions. Multiplying (2.5) by a function $v \in V$ and applying integration by parts, we obtain the usual weak form

$$(u^{n+1}, v) + \tau\epsilon(\nabla u^{n+1}, \nabla v) = \tau(f^n, v) + \big(u^n \circ y(x, t^n, t^{n+1}), v\big), \quad \forall v \in V. \tag{2.6}$$

At this point, the first advantage of the Eulerian-Lagrangian method becomes evident. The bilinear form

$$a(v,w) = (v,w) + \tau\epsilon(\nabla v, \nabla w),$$

is symmetric and positive definite. Hence solvers for symmetric systems can be used, resulting in an increase in efficiency. This is not true if the spatial derivative $b \cdot \nabla u$ is used in the discretization of (2.1).

The second advantage is the unconditional stability of the time-stepping method.

Theorem 2.1. *ELM is unconditionally stable.*

Proof. This can be shown by the following argument quickly. Taking $v = u^{n+1}$ in (2.6) gives

$$\begin{aligned}\|u^{n+1}\|^2 =&(u^n \circ y(x,t^n,t^{n+1}),u^{n+1}) - \tau\varepsilon\|\nabla u^{n+1}\|^2 + \tau(f^n,u^{n+1})\\ \leq&\|u^n \circ y(x,t^n,t^{n+1})\|\|u^{n+1}\| + \tau\|f^n\|\|u^{n+1}\|.\end{aligned}$$

Using the divergence free velocity condition (2.2) leads to

$$\|u^n \circ y(x,t^n,t^{n+1})\| = \|u^n\|,$$

i.e.,

$$\|u^{n+1}\| \leq \|u^n\| + \tau\|f^n\|.$$

Here $\|\cdot\|$ is the L^2 norm. □

Assume that Ω is a polygon and let G be a conforming, quasi-uniform grid resolving Ω. On G, define the finite element space

$$V_h^p = \{v_h \in C(\Omega)|\ v_h|_T \in \mathbb{P}_p \quad \text{for all elements } T \in G\},$$

where $\mathbb{P}_p$ is the space of all p-th order polynomials in d variables. Using V_h^p to discretize the spatial problems (2.6), several error bounds have been obtained, for relevant theoretical results, we refer to [5,7,8]. Let us first cite the following theoretical result.

Theorem 2.2. *For the pure convection problem (ϵ=0 in (2.1)), with finite element of degree p and exact time evolution, it converge with order p in $L^\infty(0,T;(L^2)^d)$ provided $u_0 \in (H^{p+1})^d$, $b \in L^\infty(0,T;(W^{1,\infty})^d)$ and $u \in H^1(0,T;(H^{p+1})^d)$.*

Higher convergence rate results are valid for some special elements on uniform grid, for example piecewise constant element with the first order convergence rate (see e.g. [8]).

The theoretical result of this method for convection-diffusion problem ($\varepsilon > 0$) can be found in [2,10] under the exact integration assumption.

Theorem 2.3. *For the convection-diffusion problem, with finite element of degree p in scheme (2.6), the convergence rate in $L^\infty(0,T;(L^2)^d)$ is $\tau + h^{p+1}$ provided*

$$u \in L^\infty\big(0,T;(H^p)^d\big), \quad \frac{\partial u}{\partial t} \in L^2\big(0,T;(H^{p-1+\theta})^d\big), \quad \frac{\partial^2 u}{\partial t^2} \in L^2\big(0,T;(L^2)^d\big),$$

and

$$b \in L^\infty\big(0,T;(W^{1,\infty})^d\big), \quad \epsilon > \epsilon_0 > 0.$$

Here $\theta = 1$ *when* $q = 2$ *and* $\theta = 0$ *when* $q > 2$ (*the convergence rate will be reduced to* $\tau + h^{p+1}/\tau$ *or* $\tau + h^p$ *for arbitrary* $\varepsilon > 0$).

Now a challenging problem is how to calculate the last integration $(u^n \circ y(x, t^n, t^{n+1}), v)$ on the right hand side of (2.6) exactly since $u^n \circ y(x, t^n, t^{n+1})$ and v are not on the same grid. All of above theoretical results are under the assumption that this integration is exactly calculated.

We now consider the evaluation of the term

$$l_2(v) = \big(u^n \circ y(x, t^n, t^{n+1}), v\big). \tag{2.7}$$

The difficulty is that while v and u^n are finite element functions on G, the composite function $u^n \circ y(x, t^n, t^{n+1})$ is not. Instead, it can be interpreted as a finite element function that has been transported along the integral lines of the velocity field b .

We discretize the ordinary differential equation (2.3) that defines y by a backward Euler scheme.

Call $y^n(x)$ the numerical approximation to $y(x, s, t^{n+1})$ at time $s = t^n$. We obtain

$$\frac{y^{n+1}(x) - y^n(x)}{\tau} = b\big(y^{n+1}(x), t^{n+1}\big).$$

Using this approximation in (2.7) gives

$$\tilde{l}_2 = \int_\Omega u^n\big(x - \tau b(x, t^{n+1})\big) \cdot v(x)dx.$$

This expression is still algorithmically intractable unless further assumptions on

$$b^{n+1} := b(\cdot, t^{n+1}),$$

are made. We therefore further simplify the problem by replacing b^{n+1} by its q-th order finite element interpolant

$$\tilde{b}^{n+1} := \mathcal{I}_q(b^{n+1}) \in (V_h^q)^d.$$

In this article we focus on the case $q = 1$. In this case, $u^n(x - \tau\tilde{b}^{n+1}(x))$ is a finite element function on an affine grid G^n which results from moving each vertex v_i of G by the vector $\tilde{b}^{n+1}(v_i)$. The product

$$\tilde{l}_2 = \int_\Omega u^n\big(x - \tau\mathcal{I}_1 b^{n+1}\big)vdx, \tag{2.8}$$

is then the product of two p-th order finite element functions on two grids G and G^n, where G^n results from G by a vertex displacement (refer to Figs. 1 and 2).

The approach described above is closely related to the "area weighting" designed for rectangle grid in [7] and also for triangular grid in [10] called "exact projection", although they did not intend for exact integration. As shown in [10], "the error caused by the approximation of the feet of the trajectories does not alter the error estimate of the exactly integrated method".

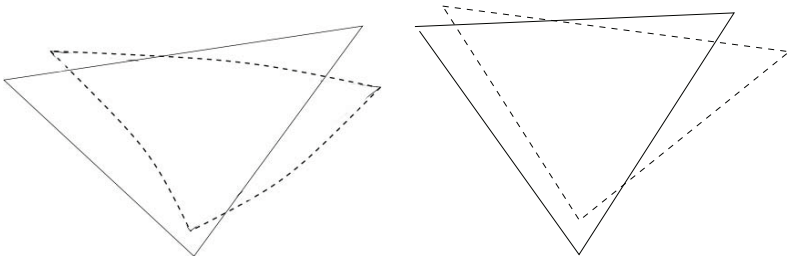

Figure 1: The element in a triangular grid G_1 (solid line) and its image of the mapping $y(x,t^n,t^{n+1})$ (dash line). The result of a general mapping is shown in left one and that of an affine mapping in right one. an affine mapping.

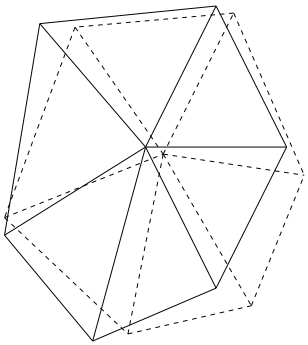

Figure 2: Two triangular grids G_1 (solid line) and G_2 (dash line) with the same topology.

3 Calculating the inner product of two finite element functions on two non-matching grids

In the previous section we have seen that the Eulerian-Lagrangian method involves integrals of the form

$$(v,w)_{\mathrm{ELM}} = \int_\Omega vwdx, \tag{3.1}$$

where v and w are both p-th order finite element functions, but on unrelated grids.

In this section, we propose an algorithm that computes integrals like (3.1) exactly. For this, note that overlaying the grids G_1 and G_2 results in a set of intersections

$$I_{ij} = T_i \cap T_j, \quad T_i \in G_1, \quad T_j \in G_2.$$

By the additivity of integration we can rewrite (3.1) as a sum over the intersections

$$\int_\Omega vwdx = \sum_{i,j} \int_{I_{ij}} vwdx.$$

The restrictions of v and w to any intersection I_{ij} are both p-th order polynomials, and hence their product is a $2p$-th order polynomial on each I_{ij}. Since I_{ij} is a convex polyhedron, the integral $\int_{I_{ij}} vwdx$ can be computed exactly by numerical quadrature, possibly after a further splitting of I_{ij} into elements.

The inner product (3.1) (and hence (2.8)) can be computed exactly if the set of intersections can be constructed. For this, an algorithm with optimal complexity is proposed in [4].

3.1 Finding the intersection of two non-matching grids

We now briefly describe the algorithm of [4] for the construction of all intersections of two grids G_1 and G_2. The algorithm works for all element types and in all space dimensions. One 2-dimensional example is shown in Fig. 3. It is of optimal time-complexity, meaning that $\mathcal{O}(N_I)$ operations are necessary to compute all intersections if N_I is the number of intersections. It is easy to construct pathological examples where every element of G_1 intersects every element of G_2. In that case, $\mathcal{O}(N_1 N_2)$ operations are necessary. Here N_i is the number of elements in G_i. However, in regular cases each element of G_1 will only intersect a constant number of elements of G_2, and vice versa. Then the number of operations will be $\mathcal{O}(N_1) = \mathcal{O}(N_2)$.

Computing the intersections of two grids remains expensive even if the algorithm is of optimal complexity. If the velocity field b depends on time then the intersections have to be recomputed at each time step. In addition to the asymptotic complexity one is therefore also interested in obtaining a low constant in $\mathcal{O}(N_I)$.

For simplicity we assume that the grids are conforming and that all elements are convex.

The algorithm consists of two parts. The first one is a mechanism that, given two elements S_1 and S_2, computes their intersection I_{12}. A simple algorithm that does this is the following.

Algorithm 3.1. Compute intersection of two elements $S_1^{\pi_i}$ and $S_2^{\textbf{wait}}$.

1. Find the intersection of all the $(n-1)$-elements of $S_1^{\pi_i}$'s boundary and all the $(n-1)$-elements of S_2^{wait}'s boundary. Record all the vertices of the intersection's boundary in set $\mathbf{V}$.
2. Add the vertices of $S_1^{\pi_i}$ in the interior of S_2^{wait} to the set $\mathbf{V}$.
3. Add the vertices of S_2^{wait} in the interior of $S_1^{\pi_i}$ to the set $\mathbf{V}$.
4. The intersection is now the convex hull of $\mathbf{V}$.

The second part is an advancing-front type algorithm that repeatedly calls the above algorithm to compute all intersections. We now describe this algorithm in two steps and show why it is optimal.

1. Find one pair of elements $S_1^{\pi_0} \in G_1$ and $S_2 \in G_2$ that intersect. That can usually be done by picking any one element $S_1^{\pi_0}$ of G_1 and then iterating over the elements of G_2 until an element intersecting $S_1^{\pi_0}$ is found.
2. Find all elements of G_2 that intersect $S_1^{\pi_0}$. This is done by a breadth-first search on G_2 starting from S_2, formally described by Algorithm 3.2. Refer to Fig. 4.

In this algorithm, elements are neighbors if they share a common boundary face.

Algorithm 3.2. Find the intersection of $S_1^{\pi_i}$ and G_2 with optimal complexity.

Require: Provide $S_2^{\pi_{j_0}}$ (has nonempty with $S_1^{\pi_i}$)

1. Initialize list L of elements on G_2 waiting for checking by $S_2^{\pi_{j_0}}$.
2. Initialize all tags T^j of the elements $S_2^{\pi_j}$ with "not checked" and $S_2^{\pi_{j_0}}$ with "checked".
3. **while** L is nonempty **do**
   Move the first one in L to S_2^{wait}.
   Find the intersection S^{int} of $S_1^{\pi_i}$ and S_2^{wait}.
   if S^{int} is nonempty **then**
      Find S_2^{wait}'s neighbors "not checked" before, add them to L and change their tags to "checked".
   end if
4. **end while**

We could now compute the set of all intersections by repeating Steps 1 and 2 for all elements $S_2^{\pi_j}$ of G_2. Step 1 has linear complexity, and hence we can only call it a constant number of times if we want to retain linear complexity of the overall algorithm. We need a faster way to get a new pair of elements $S_1^{\pi_i}$ and $S_2^{\pi_{j_0}}$ with

$$S_1^{\pi_i} \cap S_2^{\pi_{j_0}} \neq \emptyset.$$

A simple way is to pick $S_1^{\pi_{i+1}}$ as a neighbor of $S_1^{\pi_i}$ and note that it is very likely that an element of G_2 that has been checked for intersection with $S_1^{\pi_i}$ will also intersect $S_1^{\pi_{i+1}}$. If this is not true there is a cascade of fall-back strategies:

1. Pick another neighbor of $S_1^{\pi_i}$.
2. Do a brute-force search of all elements of G_2. If that fails we can conclude that $S_1^{\pi_{i+1}}$ does not intersect any elements of G_2 and mark it as done.

Unfortunately, the number of elements tested that way to find a valid seed for a given neighbor $S_1^{\pi_{i+1}}$ can still be fairly large. To lower the number, [4] used the following observation.

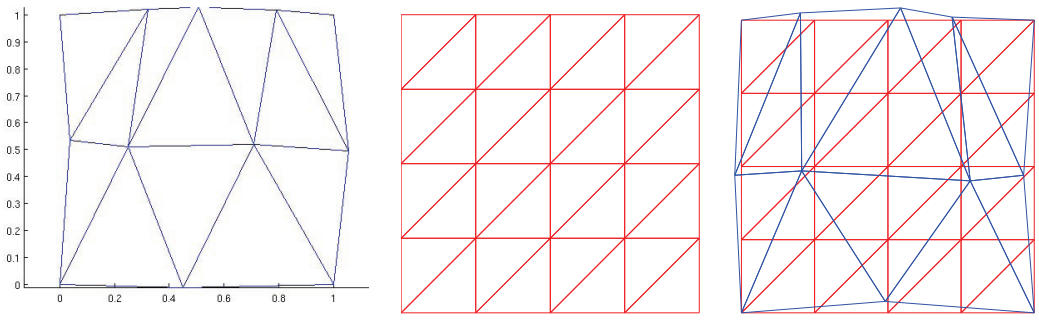

Figure 3: Two grids G_1 (left), G_2 (middle) and their intersections (right).

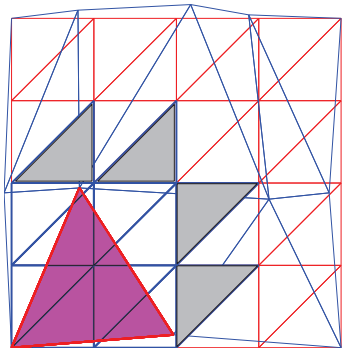

Figure 4: The elements in grid G_2 (with bold blue lines, the ones in grey are the neighbor of the ones in G_2 that have nonempty intersection with S_1 in G_1) should be checked with the element in grid G_1 (with bold red lines).

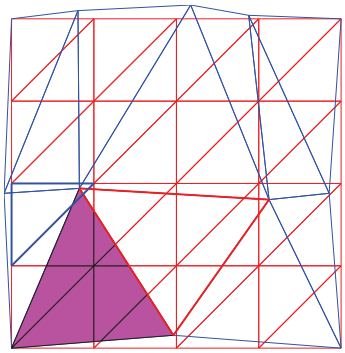

Figure 5: The first step in the processing of finding the intersection of the second element S_1^2 in G_1 (the neighbor of S_1^1, with bold blue red lines) and G_2. This step starts with the element in G_2 (with bold blue lines) having nonempty intersection with S_1^2.

Lemma 3.1. *(i) If $S_1^{\pi_i}$'s boundary $(n-1)$-element E_1 and S_2^{wait}'s boundary $(n-1)$-element E_2 have nonempty intersection, S_2^{wait} must have nonempty intersection with $S_1^{\pi_i}$'s neighbor element shared E_1 with $S_1^{\pi_i}$.*

(2) If n or more vertices of $S_1^{\pi_i}$ are located in the interior of S_2^{wait}, all $S_1^{\pi_i}$'s neighbors have no empty intersection with S_2^{wait}.

This lemma implies that we can, while computing the intersection of two elements $S_1^{\pi_i}$ and S_2^{wait}, obtain some information on whether a neighbor $\tilde{S}$ of $S_1^{\pi_i}$ also intersects S_2^{wait}. This new pair can then be used as a starting value for Step 2. Please refer to Fig. 5. The algorithm with this technique is described in Algorithm 3.3 and the whole algorithm is described in Algorithm 3.4.

Remark 3.1. A similar algorithm is provided in [3] recently. It records all the elements $S_2^{\pi_j}$ having nonempty intersection with $S_1^{\pi_i}$, and search in these $S_2^{\pi_j}$ to find $S_2^{\pi_{j_0}}$ having nonempty intersection with $S_1^{\pi_i}$'s neighbor. Compared with the algorithm to provide $S_2^{\pi_{j_0}}$ directly, it will take much more store space and cost of time.

Algorithm 3.3. Modified Algorithm 3.2 and provide $S_2^{\pi_{j_0}}$ for $S_1^{\pi_i}$'s neighbor.

Require: Provide $S_2^{\pi_{j_0}}$ (has nonempty intersection with $S_1^{\pi_i}$)

1. Initialize list L of elements on G_2 waiting for checking by $S_2^{\pi_{j_0}}$.
2. Initialize all the tags T^j of the elements $S_2^{\pi_j}$ with "not checked" and the $S_2^{\pi_{j_0}}$ with "checked".
3. Initialize all the start element tags (SST) of the neighbors of $S_1^{\pi_i}$ with "NULL".
4. **while** L is nonempty **do**
   Move the first one in L to S_2^{wait}.
   Find the intersection S^{int} of $S_1^{\pi_i}$ and S_2^{wait} providing whether S_2^{wait} has nonempty intersection with the neighbors of $S_1^{\pi_i}$.
   if S^{int} is nonempty **then**
      Find S_2^{wait}'s neighbors "not checked" before, add them to list L and change their tags to "checked".
      Find $S_1^{\pi_i}$'s neighbors having nonempty intersection with S_2^{wait} and "NULL" start element tags, and update their $SST \leftarrow S_2^{wait}$.
   end if
5. **end while**
6. **reture** SST.

Algorithm 3.4. Find the intersection of G_1 and G_2.

Require: Provide S_1^0 and S_2^0 which have nonempty intersection

1. Initialize list $L_1 = \{S_1^0\}$.
2. Initialize list $L_2 = \{S_2^0\}$.
3. Initialize all the tags T^i of the elements $S_1^{\pi_i}$ $(i > 0)$ with "not arrived" and $T^0 \leftarrow$ "arrived".
4. **while** L_1 is nonempty **do**
   $S_1^{wait} \leftarrow$ the first one in L_1 and remove the first one in L_1.
   $S_2^{start} \leftarrow$ the first one in L_2 and remove the first one in L_2.
   Call Algorithm 3.3 with S_1^{wait} and S_2^{start} and achieve SST of S_1^{wait}'s neighbors.
   Find S_1^{wait}'s neighbors not arrived at before, add them to L_1, add their SST to L_2 and change their tags $T^{neighbors}$ to "arrived".
5. **end while**

3.2 An improved algorithm

The most important technique in above algorithm is that in the step of finding the intersection of two elements, it is free to know whether they have nonempty intersection with each other's neighbor. An natural improvement, not noticed in [4], is that only S_2^{wait}'s neighbor having nonempty intersection with $S_1^{\pi_i}$ need be checked in Algorithm

3.3 with the help of the information provided by checking $S_1^{\pi_i}$ and S_2^{wait}. For example, the elements in grey shown in Fig. 4 are not necessary to be checked. Thus the algorithm can be updated as Algorithm 3.5.

Algorithm 3.5. Improved Algorithm 3.3 and provide $S_2^{\pi_{j_0}}$ for $S_1^{\pi_i}$'s neighbor.

Require: Provide $S_2^{\pi_{j_0}}$ (has nonempty intersection with $S_1^{\pi_i}$)

1. Initialize list L of elements on G_2 waiting for checking with $S_2^{\pi_{j_0}}$.
2. Initialize all the tags T^j of the elements $S_2^{\pi_j}$ with "not checked" and the $S_2^{\pi_{j_0}}$ with "checked".
3. Initialize all the start element tags SST of the neighbors of $S_1^{\pi_i}$ with "NULL".
4. **while** L is nonempty **do**
   $S_2^{wait} \leftarrow$ the first one in L and remove the first one in L.
   Find the intersection S^{int} of $S_1^{\pi_i}$ and S_2^{wait} providing whether S_2^{wait} has nonempty intersection with the neighbors of $S_1^{\pi_i}$ and whether $S_1^{\pi_i}$ has nonempty intersection with the neighbors of S_2^{wait}.
   Find S_2^{wait}'s neighbors "not checked" before and has nonempty intersection with $S_1^{\pi_i}$, add them to list L and change their tags to "checked".
   Find $S_1^{\pi_i}$'s neighbors having nonempty intersection with S_2^{wait} and "NULL" start element tags, and update their $SST \leftarrow S_2^{wait}$.
5. **end while**
6. **return** SST.

Obviously with this improvement, the two elements checked each time must have nonempty intersection. Now the complexity of this algorithm is $\mathcal{O}(N_I)$, here N_I is the number of intersections. This result corresponds to the statement that the algorithm for finding the intersection of two non-matching grids is output-sensitive or intersection-sensitive. The benefit of this improvement in complexity is obvious, however cost of time will depend on the efficiency of the operation for finding the intersection of two elements, which is usually expensive in high dimensional case. Some results of comparison are shown in Table 1. The improved algorithm saves plenty of operations for finding the intersection of two elements. We can state that more distinct benefit is found in Code in MATLAB, modified from the sample provided in [4]. This result can be explained by the efficiency of the function to check two simplex is not as high as Code in C. So it saves about 25% cost of time and the version of Code in C saves about near 10%.

Remark 3.2. The operation that finding the intersection of two given $(n-1)$-elements from different grids has been done two times in the algorithm. If this step is time-consuming, it will be valuable to avoid this duplication by storing the information in the first place. In two dimensional case, finding the intersection of two line segment is very fast. In three dimensional case, finding the intersection of two triangles seems to be so expansive that introducing this technique becomes valuable.

Table 1: The result of comparison of original and improved algorithm on a pair of grids and their regular refined ones. Code in MATLAB is provided in [4] and modified by us. Code in C is implemented by us. Test on Intel(R) Core(TM) 2 Duo CPU P8600@2.40GHz.

algorithm	number of checking	Code in MATLAB (1 time)	Code in C (50 times)
Improved	1529	1.085s	0.842s
Original	2604	1.477s	0.913s
Improved	6633	4.896s	3.423s
Original	11984	6.504s	3.735s

4 Numerical experiments

We will verify the good numerical properties of ELM with exact integration by several numerical tests. We will also show that the integration algorithm can be implemented efficiently enough to be usable even when the velocity field is time-dependent and the intersections have to be recomputed at each time step. So we will compute the intersections at each time step even when the velocity field is steady.

We have implemented the original algorithm of [4] in the C/C++ programming language and added our modification. The original article [4] contains a MATLAB implementation and we have added our modification.

4.1 Investigation of the convergence rate

We begin by studying the convergence rate of the discretization error using a benchmark problem given in [12]. Let $\Omega = [0,1]^2$ be the unit square in $\mathbb{R}^2$ and $\mathbf{b} : \Omega \to \mathbb{R}^2$ the velocity field

$$b = (y, -x).$$

We consider the general linear convection-diffusion equation

$$\frac{\partial u}{\partial t} + b \cdot \nabla u - \epsilon \Delta u = 0, \quad \text{on } \Omega, \tag{4.1}$$

and zero boundary condition.

If the initial condition at $t = 0$ is chosen as

$$u_0(x,y) = \exp\Big[-\frac{\hat{x}^2 + \hat{y}^2}{2\lambda^2}\Big],$$

the exact solution of (4.1) is given by

$$u(x,y,t) = \frac{\lambda^2}{\lambda^2 + 2\epsilon t} \exp\Big[-\frac{\hat{x}^2 + \hat{y}^2}{2(\lambda^2 + 2\epsilon t)}\Big].$$

Here and above we have set

$$\hat{x} = x - x_0 \cos t - y_0 \sin t \quad \text{and} \quad \hat{y} = y + x_0 \sin t - y_0 \cos t.$$

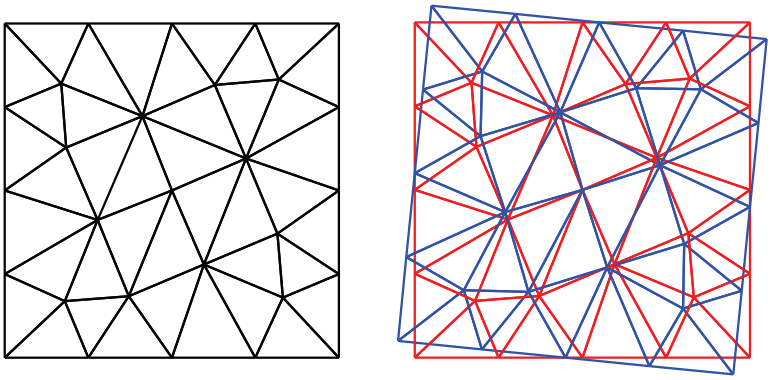

Figure 6: Left: coarse grid. Right: coarse grid together with grid moved one backward Euler step of size $\tau = 0.1$ along the integral lines of b.

The numbers λ, x_0, and y_0 are constants and in this example we have used

$$\lambda = \frac{1}{8} \quad \text{and} \quad (x_0, y_0) = \Big(-\frac{1}{2}, 0\Big).$$

We discretize (4.1) and the flow equation (2.3) in time using a backward Euler scheme. For discretization in space we use first-order finite elements on the grid given in Fig. 6. In this example, if backward Euler scheme is used to solve the equation of characteristic foot in scheme (2.5), our algorithm will generate exactly the same as the exact integration. The coarsest grid in our numerical study and its deformed grid are shown in Fig. 6.

As described in Section 2, the linear functional

$$\tilde{l}_2 = \int_\Omega u^n(x - \mathcal{I}_1 b^{n+1}) v dx, \tag{4.2}$$

now involves two piecewise linear functions on two different grids. Our algorithm presented in Section 3 can integrate this exactly. We now run our algorithm for 200 time steps of time step size $\tau = 10^{-6}$ on various levels of global grid refinement. For each run we compute the error

$$\|u_h - u\|_{L^\infty([0,T],L^2(\Omega))} = \max_{0\le n\le 200} \|u_h(t^n) - u(t^n)\|_{L^2(\Omega)}.$$

The results are plotted in Fig. 7.

Since we integrate the term (4.2) exactly, from Theorem 2.3 we expect a convergence order of $\mathcal{O}(\tau) + \mathcal{O}(h^2)$. One does indeed observe the quadratic dependence on the mesh size.

To also check the dependence of the error on the time step size τ we redo the computations, this time on a fixed time interval $[0, 1]$. We fix a diffusion coefficient $\epsilon = 10^{-3}$ and first select a time step size of $\tau = 0.5h$. From Theorem 2.3 one would then expect linear convergence of the discretization error.

As a second test we set $\tau = h^2$ Then we expect a quadratic decrease of the discretization error as a function of the mesh size. This is confirmed numerically, as can be seen from Fig. 8.

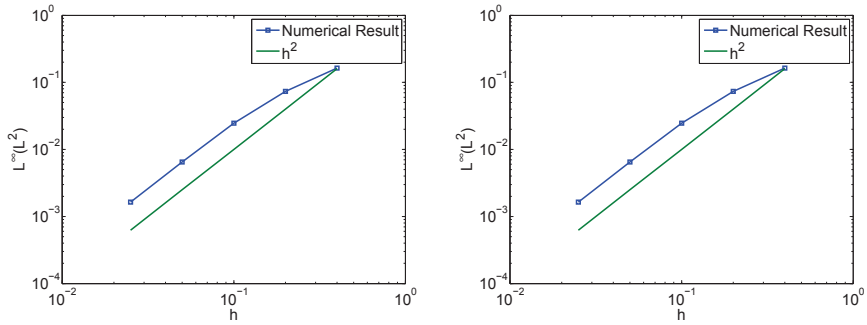

Figure 7: $L^\infty(L^2)$ norm of $u-u_h$ on triangular grid with space size $h=0.4,0.2,0.1,0.05,0.025$, time step $\tau=10^{-6}$ in 200 steps of time marching. The diffusion coefficient is 10^{-3} (left) and 10^{-6} (right).

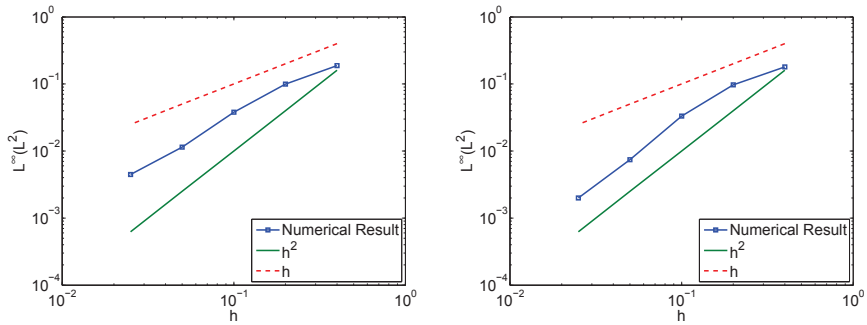

Figure 8: $L^\infty(L^2)$ norm of $u-u_h$ on triangular mesh with space size $h=0.4,0.2,0.1,0.05,0.025$, time step $\tau=0.5h$ (left) and $\tau=h^2$ (right) in time interval $[0,1]$. The diffusion coefficient is 10^{-3}.

4.2 Comparison with other numerical quadratures

In the next example we compare our method to an approach where the integrals $\tilde{I}_2$ are approximated by Gaussian quadrature rules on the elements of G. It was reported that such schemes can achieve optimal convergence order if the finite element solution is continuous [5]. However it is only numerical study and cannot be proved theoretically. Here we show that achieving optimal convergence rate by numerical quadrature is not true in general and stress the importance of accurate integration.

We use again the continuous problem setting of the previous section. However, we focus on the pure convection case, that is we set $\epsilon=0$. Then the space of solutions is $L^2(\Omega)$, which we discretize using piecewise constant finite elements. The analysis of convergence rate can be found in [8]. The optimal convergence rate is expected as $\mathcal{O}(\tau)+\mathcal{O}(h)$, however this is only proved rigorously for the one-dimensional case.

The general processing of ELM using quadrature rule is

$$\left(u^n\circ y(x,t^n,t^{n+1}),v\right)=\int_{\Omega_v}u^n\circ y(x,t^n,t^{n+1})\cdot v\approx\sum_i u^n\circ y(x_i,t^n,t^{n+1})\cdot v(x_i)W_i,$$

where Ω_v is the support set of test function v and W_i is the weight of quadrature point. When u^n is not smooth enough, discontinuous in this case, the quadrature rule will not capture the right value of solution from the neighboring element. As shown in Fig. 9,

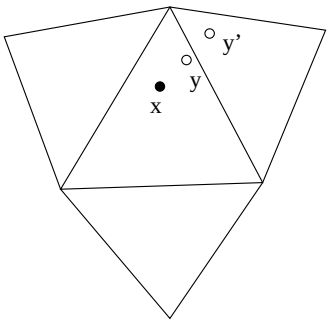

Figure 9: The location of quadrature point's characteristic foot, y' when time step k is large and y when k is small.

if time step k is not very large, no quadrature point's images under flow mapping y can arrive at the region of the neighboring element, where the convection comes from. In the worst case, the finite element solution will always be the same as initial solution. High order quadrature rule has more quadrature points and some of their images can arrive at neighbor element, but it still will not catch the exact information of convection.

A series of results are shown in Fig. 10. In small time step case, using quadrature rule to calculate the integration may lead to a totally wrong solution (refer to 4 quadrature points case, the numerical solution is always the same as initial one). Usual suggestion for this problem are using more quadrature points to improve the results. That's right but it has a slow improvement and will take terrible cost of time especially in weak regularity case. Another choice is to take large time step. It also has inherently shortcoming that lose some property of solution. For example, the peak point of numerical solution runs far away from the trajectory of the exact peak point.

4.3 Comparison of time cost

The big advantage of this algorithm is to avoid the huge time cost of solving a lot of the equations of characteristic feet in high order quadrature rule. Due to in each intersection of two grids, the low order quadrature rule can approach the exact integration. It requires an assumption that the algorithm to find the intersection of two grids and the relationship of them should be not time consuming. Otherwise it cannot be used in evolution problem with time dependent velocity field. The result of time cost in the numerical examples in last subsection is shown in Table 2. It can attain the smallest

Table 2: The result of time cost and $L^\infty(L^2)$ norm of error in last example (on the mesh $h = 0.05$, time step $k = 0.2h$ and the time interval $[0, \pi/4]$). Test on Intel(R) Core(TM) 2 Duo CPU P8600@2.40GHz. # means number.

# of points	4	6	12	33	73	new algorithm
Time cost (s)	1.67	2.26	3.92	8.69	19.13	9.61
$L^\infty(L^2)$	0.298190	0.255570	0.132275	0.101159	0.0523266	0.0514938

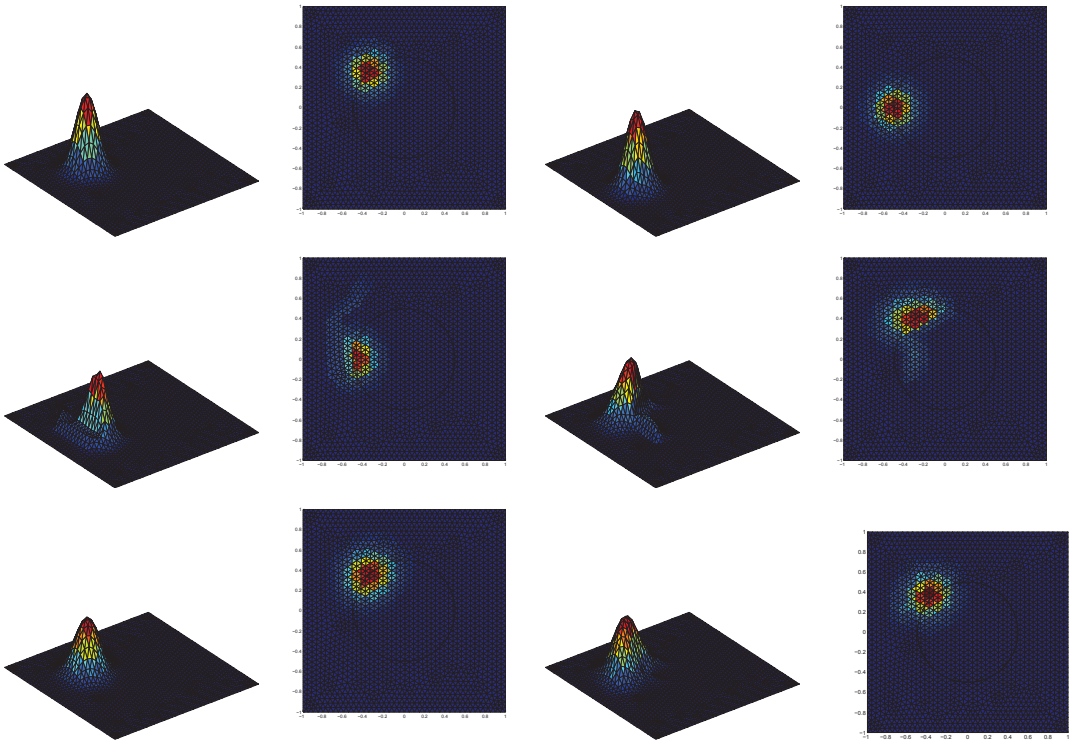

Figure 10: The result of the pure convection problem at $t = \pi/4$. Shown are the exact solution (row 1, left), the solution with $\tau = 0.2h$ using 4 quadrature points (row 1, right), the same with 6 quadrature points (row 2, the left pair), 12 quadrature points (row 2, the right pair) and the exact integration achieved by our algorithm (row 3, the first pair), and the solution with $\tau = 2h$ using 6 quadrature points (row 3, the second pair). The circle is the trajectory of solution's peak point.

error with not much time cost. So this algorithm can be applied in time dependent velocity field problem and provide better accuracy.

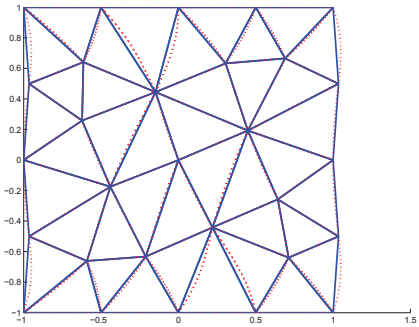

Figure 11: The deformed grid (dash) and its approach by lines (solid) with time step $\tau = 0.1$, $h = 0.04$ and velocity field $(\sin \pi y^2, 0)$.

4.4 Inexact Integration

As we pointed out before, this scheme cannot achieve the exact integration in general case. Here one example is used to show the effort of the approximation of the deformed grid on the convergence rate.

The velocity field

$$b = (\sin \pi y^2, 0),$$

and the solution

$$u = u_0(x - t \sin \pi y^2, y),$$

satisfy the pure convection equation on $[-1,1] \times [-1,1]$. Here u_0 is the initial solution and the grids are shown in Fig. 11. The convergence rate is shown in Fig. 12 and still $\mathcal{O}(h)$ for spatial discretization using piecewise constant element. It confirms the statement that "the error caused by the approximation of the feet of the trajectories does not alter the error estimate of the exactly integrated method" in [10].

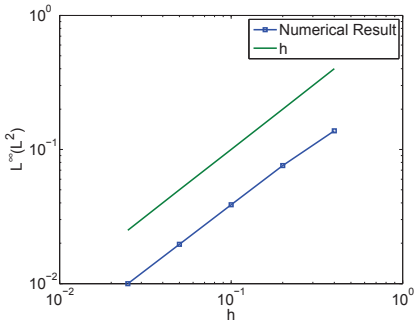

Figure 12: $L^\infty(L^2)$ norm of $u - u_h$ on triangular mesh with space size $h = 0.4, 0.2, 0.1, 0.05, 0.025$, time step $\tau = 0.1h$ in time interval $[0,1]$ with velocity field $(\sin \pi y^2, 0)$.

5 Concluding remarks

In this special application of ELM, there is only one original grid and its deformed grid. So if the time step is small enough, i.e., the vertices of a element will not go out of the patch where they stay at, the number of pairs of elements from two different grids having nonempty intersection can be estimated by CN. Here C is the maximum number of elements in the patch and N is the number of elements in the grid. For

Table 3: Cost of time in finding the intersection of the grids used in last section using the new improved algorithm (the ones before the parentheses) and special algorithm (the ones in the parentheses) for ELM (50 operations). Test on Intel(R) Core(TM) 2 Duo CPU P8600@2.40GHz and time step $\tau = 10^{-4}$. # means number.

# of elements	40	160	640	2560	10240
# of intersections	190	776	3148	12692	50890
Cost of time	0.11(0.10)	0.33(0.32)	1.15(1.13)	4.35(4.37)	19.04(19.05)

the shape regular grid, C is finite. Of course, this is a very rough up-bound. In the example shown in Table 3, C seems to be smaller than 5 and the maximum number of elements in the patch for that grid is 15.

One possible improvement of the whole algorithm in this application is to provide $S_2^{\pi_{j_0}}$ for $S_1^{\pi_i}$ by the image of $S_1^{\pi_i}$ under the flow mapping. Of course, it is only true when the image of $S_1^{\pi_i}$ has the none-empty intersection with $S_1^{\pi_i}$. This can be easily satisfied when time step is small enough. In practice, it cannot show much obvious help in time of cost (shown in Table 3). It can be understood by the fact that we can provide $S_2^{\pi_{j_0}}$ for $S_1^{\pi_i}$ without any additional cost. However, it still can convince us the above whole algorithm will not destroy the property of ELM in parallel computation.

Now ELM using accuracy integration is available and effective. This algorithm can achieve the optimal convergence rate, which has been proved in analysis with the exact integration. The most important advantage of the algorithm now is that it does not depend on the spatial dimension of the physical domain and the type of finite elements used in discretization. Moreover the advantage of this type of ELM becomes more prominent when the regularity of solution space becomes weaker. For this reason, in our future works, we will especially apply this algorithm for using ELM with $H(\mathbf{curl})$, $H(\mathbf{div})$ or L_2 finite element discretization spaces.

Acknowledgments

The author wish to thank Dr. Oliver Sander for his many helpful discussions about the relevant algorithms in the paper and for his great help for improving the exposition of the paper. The author also wish to thank Professors Jinchao Xu and Pingwen Zhang for helpful discussions and encouragement. Authors would like to thank referees for valuable comments for improving this paper.

References

[1] M. De Berg, O. Cheong, M. Van Kreveld and M. Overmars, Computational Geometry: Algorithms and Applications, Springer Verlag, 2008.

[2] J. Douglas Jr and T. F. Russell, *Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures*, SIAM J. Numer. Anal., 5 (1982), pp. 871–885.

[3] P. E. Farrell and J. R. Maddison, *Conservative interpolation between volume meshes by local Galerkin projection*, Comput. Methods Appl. Mech. Eng., accepted, 2010.

[4] M. J. Gander and C. Japhet, *An algorithm for non-matching grid projections with linear complexity*, Lect. Notes Comp. Sci. XVIII, (2009), pp. 185–192.

[5] J. Jia, X. Hu, J. Xu and C. Zhang, *Effects of integrations and adaptivity for the Eulerian-Lagrangian method*, J. Comput. Math., accepted.

[6] K. W. Morton and R. B. Kellogg, Numerical Solution of Convection-Diffusion Problems, Chapman & Hall London, 1996.

[7] K. W. MORTON, A. PRIESTLEY AND E. SÜLI, *Stability of the Lagrange-Galerkin method with nonexact integration*, RAIRO Modél, Math. Anal. Numér., 4 (1988), pp. 625–653.
[8] K. W. MORTON AND E. SÜLI, *Evolution-Galerkin methods and their supraconvergence*, Numer. Math., 3 (1995), pp. 331–355.
[9] O. PIRONNEAU, *On the transport-diffusion algorithm and its applications to the Navier-Stokes equations*, Numer. Math., 3 (1982), pp. 309–332.
[10] A. PRIESTLEY, *Exact projections and the Lagrange-Galerkin method: a realistic alternative to quadrature*, J. Comput. Phys., 2 (1994), pp. 316–333.
[11] T. F. RUSSELL AND M. A. CELIA, *An overview of research on Eulerian-Lagrangian localized adjoint methods (ELLAM)*, Adv. Water Resour., 8-12 (2002), pp. 1215–1231.
[12] D. XIU AND G. E. KARNIADAKIS, *An algorithm for non-matching grid projections with linear complexity*, J. Sci. Comput., 1 (2002), pp. 585–597.