LEoPART: a particle library for FEniCS

Jakob M. Maljaars\textsuperscript{a}, Chris N. Richardson\textsuperscript{b}, Nathan Sime\textsuperscript{c}

\textsuperscript{a}Environmental Fluid Mechanics, Faculty of Civil Engineering and Geosciences, Delft University of Technology, Stevinweg 1, 2600 GA Delft, The Netherlands
\textsuperscript{b}University of Cambridge BP Institute, Madingley Road, Cambridge, CB3 0EZ, UK
\textsuperscript{c}Department of Terrestrial Magnetism, Carnegie Institution for Science, Washington, DC, USA

Abstract

This paper introduces LEoPART, an add-on for the open source finite element software library FEniCS to seamlessly integrate Lagrangian particle functionality with (Eulerian) mesh-based finite element (FE) approaches. LEoPART - which is so much as to say: 'Lagrangian-Eulerian on Particles' - contains tools for efficient, accurate and scalable advection of Lagrangian particles on arbitrary polyhedral meshes. In addition, LEoPART comes with several projection operators for exchanging information between the scattered particles and the mesh and vice versa. These projection operators are based on a variational framework, which allows extension to high-order accuracy. In particular, by implementing a dedicated PDE-constrained particle-mesh projection operator, LEoPART provides all the tools for diffusion-free advection, while simultaneously achieving optimal convergence and ensuring conservation of the projected particle quantities on the underlying mesh. A range of numerical examples that are prototypical to passive and active tracer methods highlight the properties and the parallel performance of the different tools in LEoPART. Finally, future developments are identified. The source code for LEoPART is actively maintained and available under an open source license at \url{https://bitbucket.org/jakob_maljaars/leopart}.

Keywords: particle-in-cell, Finite Elements, PDE-constrained optimization, particle tracking, Software, FEniCS

1. Introduction

Passive and active tracer methods find applications in a versatile range of engineering areas such as geophysical flows and environmental fluid mechanics \cite{1,2,3,4}, experimental fluid mechanics \cite{5,6,7}, and bio-medical applications \cite{8,9}, to name a few. In passive tracers methods, the Lagrangian particle motion is fully determined by the carrier flow and there is no feedback mechanism from the particles to the carrier flow. Such a feedback mechanism between the tracer particles and the carrier fluid is, however, typical to active tracer methods, in which particles actively affect the flow properties of the carrier fluid and vice versa. To
capture this interaction between the particles and the flow in numerical models, physical information which
is carried by the tracer particles is used to estimate one or more additional source terms in the governing
fluid equations. In a discrete setting, this typically requires the reconstruction of mesh fields from the
scattered particle data when the fluid flow equations are solved using a mesh-based approach, such as finite
difference (FD), finite volumes (FV), or finite elements (FE). Application examples of active tracer methods
include, among many others, the modelling of turbulent (reacting) flows \cite{10, 11}, and mantle convection
problems \cite{4, 12, 13, 14}. Alternatively, the particle information can be used to solve the advective part
of physical transport phenomena, resulting in so-called particle-mesh operator splitting schemes, see, e.g.,
\cite{15, 16}, and the earlier work by Maljaars et al. \cite{17, 18, 19} from which LEOPART has evolved. Eliminating
artificial dissipation by using Lagrangian particles for the discretization of the advection operator primarily
motivates such methods, which are particularly promising for simulating advection dominated flows \cite{17, 18}
or free-surface flows \cite{14, 20}.

In all the aforementioned methods and applications, particle-based and mesh-based discretization tech-
niques essentially become intertwined. To render such a combination of Lagrangian particle methods in
conjunction with mesh-based FD, FV, or FE solvers tractable for simulating practical engineering problems,
a suite of dedicated, flexible and efficient tools is indispensable.

The open source library LEOPART which is presented in this paper provides such a toolbox by inte-
grating particle functionality in the open source FE library FEniCS \cite{21}. LEOPART - which stands for
‘Lagrangian-Eulerian on Particles’ - contains utilities for efficiently and flexibly advecting and tracking a
set of user-defined particles on arbitrary polyhedral meshes. In addition, LEOPART contains a suite of tools
for projecting scattered particle data onto the mesh and \textit{vice versa} in a high-order accurate, efficient and
physically sound manner. These features render LEOPART unique in its kind in that it is one of the few,
if not the only, open source package embedding Lagrangian particles functionality in a mature open source
FE library. The resulting combination of LEOPART and FEniCS is particularly suited for application to
flow problems involving active or passive tracers, or to implement particle-mesh operator splitting schemes,
as will be demonstrated by various numerical examples throughout.

The paper is structured as follows. Section 2 gives some background information on the encompassing
FEniCS library, and provides a helicopter view of LEOPART. Section 3 describes the implementation of
particles, as well as the advection and tracking of particles in LEOPART. Section 4 details the available
particle-mesh interaction strategies, paying particular attention to the PDE-constrained particle-mesh in-
teraction in Section 4.1.3, which enables the reconstruction of conservative mesh fields from a set of moving
particles. Section 5 illustrates some example applications, meanwhile paying attention to the performance
and scaling properties of LEOPART. Section 6 closes the paper by presenting conclusions and providing an
outlook on future developments.
2. Implementation in FEniCS

2.1. A primer on FEniCS

FEniCS is a Finite Element (FE) framework, written in C++, with a Python interface. One of the major challenges of writing an FE code is that the computation which needs most user configuration is that of the local element tensor, which lies at the innermost part of the assembly loop. The local element tensor describes the matrix entries on each individual element, and relates to the physical equations of the system. FEniCS solves this problem by allowing the user to write these equations in a Domain Specific Language (DSL), which is then automatically compiled to C code to be called by the assembly loop.

In addition to simplifying the construction of the local element tensor, FEniCS makes it easier to run in parallel using MPI. Running a DOLFIN code with MPI can be as simple as `mpirun -n 32 python3 demo.py`. The mesh will be partitioned into 32 chunks, and the problem will be distributed across 32 processes for this job.

FEniCS consists of several components:

- Unified Form Language (UFL) - providing the DSL component
- FEniCS Form Compiler (FFC) - which compiles the DSL into C code
- FIAT - the finite element tabulator
- DOLFIN - the main C++/Python package, which integrates I/O, assembly and solvers

The following Python script shows how the DOLFIN interface can simplify the expression of an FE problem and computation of its solution. Consider the weak form of the Poisson equation defined on the unit square $\Omega := (0,1)^2$: find $u \in V$ subject to homogeneous Dirichlet boundary conditions, where $V$ is the appropriate solution space, such that

$$\int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} v \sin(x) \, d\Omega \quad \forall \, v \in V.$$  

Python code

```python
from dolfin import *

mesh = UnitSquareMesh(20, 20)
V = FunctionSpace(mesh, "Lagrange", 1)
u, v = TrialFunction(V), TestFunction(V)
a = inner(grad(u), grad(v))*dx
f = Expression("sin(x[0])", degree=2)
L = f*v*dx
u = Function(V)
bc = DirichletBC(V, 0.0, "on_boundary")
solve(a==L, u, bc)
```

Calling the `solve()` method runs the form compiler (FFC) and compiles the symbolic expressions into C code, which is compiled and loaded into memory. A global matrix equation is then assembled using this
generated code for the local element tensor, and finally an LU solve is called to solve the resulting system of equations. Whilst this example is simple and compact, many options exist to expand each part of the problem, for example by applying Dirichlet boundary conditions, or assembling matrix and vector separately, and choosing more sophisticated solvers.

For larger problems, it is important to run in parallel using MPI. Mesh partitioning is performed using PT-SCOTCH or ParMETIS, and there is support for the HDF5 file format, which allows parallel access of large datasets. Third party libraries are used throughout, wherever possible: PETSc is the linear algebra backend of choice, with a large selection of parallel solvers available.

FEniCS is an open source package, and is available for various platforms. The latest information can be found on the project website [www.fenicsproject.org](http://www.fenicsproject.org).

### 2.2. LEOPART code structure

LEOPART is built on top of the FEniCS package, and adds new concepts for the advection of particles on polygonal meshes, and the interaction between particles and the mesh. A central paradigm in the design of LEOPART is that it serves as an add-on to FEniCS, using the existing FEniCS tool chain wherever possible. As a result all the FEniCS functionality remains available for the user. In particular, LEOPART is designed such that it seamlessly integrates with the mesh partitioning in FEniCS facilitating parallelism using MPI.

To provide a fast and user friendly suite of tools, LEOPART wraps C++ code in Python using `pybind11` for the computationally demanding parts such as the particle advection and the matrix assembly. Particle pre-processing and post-processing is done in Python. Fig. 1 provides an overview of the different core components in LEOPART.

![Figure 1: Code structure LEOPART.](https://github.com/pybind/pybind11)

In the remainder of this paper, we essentially discuss the different components from Fig. 1. Particular attention is paid to the particles class, the advection of the particles, and the tools for the interaction between the particles and the mesh via \( \ell^2 \) projections or PDE-constrained projections.

---

1https://github.com/pybind/pybind11
As an aside, we note that LEOPart also contains a Stokes solver, implementing the H(div) conforming hybridized discontinuous Galerkin (HDG) formulation from Rhebergen & Wells [22, 23]. The corresponding variational forms are found in \texttt{FormsStokes.py} and the solver is available in \texttt{StokesStaticCondensation.h}. Implementation details of this solver are beyond the scope of this paper.

3. Particle Functionality

This section explains the implementation of particles in LEOPart and discusses the particle advection and particle tracking strategy on polyhedral meshes.

3.1. Particle initialization

The \texttt{particles} class forms the backbone for dealing with the Lagrangian particles in LEOPart. Operations such as particle advection and the particle-mesh projections require as input an instance of this \texttt{particles} class, see Fig. 1.

Each particle is assumed to have at least one property attached to it, viz. a spatial coordinate, which henceforth is denoted by $x_p$ for a single particle. Moreover, it is presumed that particles always live in a spatial domain, denoted $\Omega$, so that the particle coordinate set is defined as

$$X_t := \{x_p(t) \in \Omega\}_{p=1}^{N_p}$$

with $N_p$ the total number of particles. For notational convenience, we also will make use of the index set of particles and the index set of particles hosted by cell $K$, at a fixed time instant $t$, which are defined as

$$S_t := \{p \in \mathbb{N} : x_p(t) \in X_t\}$$

$$S^K_t := \{p \in \mathbb{N} : x_p(t) \in K, x_p(t) \in X_t\}$$

Whereas carrying a spatial coordinate might be sufficient for passive particle tracing, additional properties, such as density, concentration, or momentum values, need to be attached to the particles for active particle tracing. For a scalar and vector valued property, such particle quantities are defined as

$$\Psi_t := \{\psi_p(t) \in \mathbb{R}\}_{p=1}^{N_p}$$

$$\mathcal{V}_t := \{v_p(t) \in \mathbb{R}^d\}_{p=1}^{N_p}$$

respectively, where $d = 1, 2, 3$ is the spatial dimension.

The set of Lagrangian particles which is formed by the coordinate set $X_t$, and an arbitrary number and ordering of scalar and/or vector valued particle quantities is used to instantiate the \texttt{particles} class in LEOPart. For example, instantiating the \texttt{particles} class on a unit square mesh from a user-defined coordinate array, together with arrays for a scalar- and a vector-valued property is as straightforward as
from dolfin import UnitSquareMesh
from leopart import particles
import numpy as np

msh = UnitSquareMesh(2, 2)

# Create 4 particles in 2D
xp = np.random.rand(4, 2)
psip = np.zeros(xp.shape[0])
vp = np.zeros(xp.shape)

# Instantiate particles
p = particles(xp, [psi_p, v_p], msh)

For the sake of generality it is noted that the ordering and the length of the list with the particle quantities, i.e. [psi_p, v_p] in the above example is arbitrary.

3.2. Particle advection

Three different particle advection schemes are currently supported by LEO Part. These advection schemes solve the system of ODEs: given a vector-valued velocity field \( \mathbf{a}_h \), solve \( \forall \ p \in S_t \)

\[
\frac{dx_p}{dt} = \mathbf{a}_h(x_p, t) \\
\frac{d\psi_p}{dt} = 0, \\
\frac{dv_p}{dt} = 0,
\]

where a particle can carry an arbitrary number of scalar- and/or vector-valued quantities that will stay constant during the particle advection stage. The \texttt{advect\_particles} class solves Eq. (7) with a first-order accurate Euler forward method, and the two and three stage Runge-Kutta methods are available via the \texttt{advect\_rk2} and the \texttt{advect\_rk3} classes, respectively. The two multi-stage Runge-Kutta advection schemes inherit from the \texttt{advect\_particles} class, and a typical constructor for the latter reads

```cpp
// Constructor advect\_particles
advect\_particles(particles& P, FunctionSpace& U, Function& uhi,
                 const MeshFunction<std::size_t>& mesh\_func);
```

This snippet shows that the particle advection classes require a \texttt{particles} instance, a velocity field specified in the \texttt{Function} and its corresponding \texttt{FunctionSpace}, and a \texttt{MeshFunction} for marking the boundaries, see Section 3.3. A complete overview of all the particle advection constructors is found in \texttt{advect\_particles.h}

Imperative for both the particle advection, as well as the particle-mesh interactions later on, is the evaluation of mesh fields at a potentially large number of points inside the domain. In order to do so, it
must be known in which cell a particle resides, for which two options are available at a meta-level. The first option is that each particle carries a reference to its hosting cell in the mesh. As soon as a particle crosses a cell boundary, this particle-to-cell reference is updated. The second option is to consider a cell as a bucket filled with particles. Rather than a particle keeping track of its hosting cell, the bookkeeping is done at the cell level, i.e. each cell contains a list of particles. When a particle leaves a cell, the particle is removed from the cell’s ‘particle bucket’, and added to the receiving cell’s particle bucket.

The latter method is used in LEOPart as this allows efficient evaluation of a mesh-field at the particle positions. The expansion coefficients for the mesh-field are evaluated only once per cell, and are subsequently re-used for all particle positions within that cell. In parallel computations with distributed memory, the mesh partitioning which is taken care of by FEniCS, provides a convenient structure to store the particle data on the different processes. From a load balance perspective, such an approach is particularly suited for cases in which particles are more or less uniformly distributed over the spatial domain.

A second challenge, specifically related to the particle advection, is to efficiently keep track of the hosting cell for the Lagrangian particles in the unstructured polyhedral mesh. In the literature several procedures have been developed with this purpose. For example: superposition of a coarse Cartesian mesh onto the unstructured mesh [24], the tetrahedral walk method [25], or methods based on barycentric interpolation [26]. An alternative method is the convex polyhedron method [27], which assumes that the mesh consists of convex polyhedral cells. This indeed is the case for the simplicial cells used in FEniCS. For each facet in the mesh, the midpoint, the unit normal, and the connectivity of the facet are pre-computed. Concerning the connectivity, a facet has two neighboring cells for facets internal to the mesh, or only one neighboring cell for exterior boundary facets and internal facets which are located on processor boundaries. From the perspective of a mesh cell, indicated by $K$, the sign of the facet unit normals is adapted so as to make sure that they are always outwardly directed.

![Figure 2: Particle tracking using the convex polyhedron strategy.](image-url)
Algorithm 1 Convex polyhedron particle tracking: pseudo-code for a single particle initially located in a cell \( K \), using Euler forward for the particle advection.

1: \( k \leftarrow 0 \)
2: \( \Delta t^{(k)} \leftarrow \Delta t \)
3: \( x_p^{(k)} \leftarrow x_p(t) \)
4: Hosting cell: \( K \)
5: while \( \Delta t^{(k)} > 0 \) do
6: \( k \leftarrow k + 1 \)
7: Time to facet intersection: \( \Delta t_i = b_i/(a(x_p^0, t) \cdot n_{K_i}) \), with \( i \) the indices of the neighboring cells.
8: Minimum, yet positive time: \( \Delta t_{i_{\text{min}}} = \min\{\max(0, \Delta t_i)\} \), with \( i \) the indices of the neighboring cells
9: if \( \Delta t_{i_{\text{min}}} > \Delta t^{(k)} \) then
10: Particle remains in cell \( K \)
11: \( x_p^{(k)} \leftarrow x_p^{(k-1)} + a(x_p^0, t) \cdot \Delta t^{(k)} \)
12: \( \Delta t^{(k)} \leftarrow 0 \)
13: else
14: Push particle to facet
15: \( x_p^{(k)} \leftarrow x_p^{(k-1)} + u(x_p^0, t) \cdot \Delta t_{i_{\text{min}}} \)
16: \( \Delta t^{(k)} = \Delta t^{(k-1)} - \Delta t_{i_{\text{min}}} \)
17: if Facet has two neighboring cells then
18: Update hosting cell index: \( K \leftarrow K_{i_{\text{min}}} \)
19: else
20: Apply boundary condition, see Section 3.3
21: end if
22: end if
23: end while
24: \( x_p(t + \Delta t) \leftarrow x_p^{(k)} \)

The convex polyhedron particle tracking then proceeds as follows for a particle \( p \), which at time \( t \) is located at \( x_p(t) \) within cell \( K_0 \), see Fig. 2 for a principle sketch. Assume the particle has a velocity \( a(x_p, t) \) and the time step used for advecting the particle is \( \Delta t^{(0)} = \Delta t \), where the use of a superscript will become clear shortly. As the first step in the particle tracking algorithm, the time to intersect the i-th facet of element \( K_0 \) is computed as \( \Delta t_i = b_i/(a(x_p^0, t) \cdot n_{K_i}) \), with \( b_i \) the orthogonal distance between the particle and the facet with index \( i \). Next, the minimum, yet positive, time to intersection is computed as \( \Delta t_{i_{\text{min}}} = \min\{\max(0, \Delta t_i)\} \), with \( i \) the indices of the neighboring cells. If \( \Delta t_{i_{\text{min}}} > \Delta t^{(k)} \) the particle is pushed to its new position using timestep \( \Delta t^{(k)} \), and the time step is terminated by setting \( \Delta t^{(k+1)} = 0 \). If \( \Delta t_{i_{\text{min}}} < \Delta t^{(k)} \), the particle \( p \) is pushed to the facet intersection \( x_p^{(k)} \) using \( \Delta t_{i_{\text{min}}} \), and the hosting cell is updated to facet \( i_{\text{min}} \), that is the facet with the index corresponding to \( \Delta t_{i_{\text{min}}} \). Furthermore, the time step
is decremented to $\Delta t^{(k+1)} = \Delta t^{(k)} - \Delta t_{\text{min}}$, after which the particle tracking continues until the time step for a particle has zero time remaining. Algorithm 1 contains the pseudo-code for the convex polyhedron particle tracking, using an Euler method for the particle advection.

The convex polyhedron procedure comes with a number of advantages, also pointed out in [27]. First of all, it is applicable to arbitrary polyhedral meshes, both in two and three spatial dimensions, on the premise that cells are convex. Secondly, by marking the facets on the boundary of the domain, it is straightforward to detect if, when, and where a particle hits a specific boundary. This feature is useful when dealing with external boundaries, as well as internal boundaries, with the latter resulting from the mesh partitioning in parallel computations. Finally, the fraction of the time step spent in a certain cell is explicitly known in the convex polyhedron method. This can facilitate the updating of the particle velocity along its trajectory [28], although it is not further pursued in the code in its present form.

3.3. Boundary conditions at particle level

Apart from enforcing the boundary conditions on the background mesh, modifications at the particle level are also required when a particle hits a specific boundary. This event is detected when a particle is pushed to a facet having only one neighboring cell, see Line 20 in Algorithm 1. The following particle boundaries are supported in LEoPART, see for details advect_particles.h

```c++
// enum for external facet types
enum class facet_t : std::uint8_t {
    internal,
    closed,
    open,
    periodic
};
```

The user can mark the different parts of the boundary using a MeshFunction, where this mesh function is passed to the particle advection class, see, for instance, the unit tests in test_2d_advect.py. The implementation of the different particle boundary conditions is briefly discussed below.

3.3.1. Periodic boundaries

When a particle crosses a facet which is marked as a periodic boundary, it should reappear at the opposing side of the domain. To implement this, facets on a periodic boundary, need to be matched against facets at the opposing side of the domain. This is taken care of in the advection class when the boundary is marked as a periodic boundary via a MeshFunction, and the coordinate-limits of the opposing boundaries are specified pairwise by the user. To illustrate this, a bi-periodic unit square domain is marked in LEoPART, when using the forward Euler particle advection, as
Full code examples of how periodic boundaries are applied in 2D and 3D are found in `TaylorGreen_2D.py` and `TaylorGreen_3D.py`.

### 3.3.2. Open boundaries and particle insertion/deletion

At boundaries that are marked as open, particles either escape or enter the domain. When a particle escapes through an open boundary facet, it simply is deleted from the list of particles. Inflow boundaries, however, are less straightforward since new particles are to be created. This is done via the `AddDelete` class, which also allows a user to keep control over the number of particles per cell.

`AddDelete` takes as arguments the particle class, a lower and an upper bound for the number of particles per cell, and a list of FEniCS functions which are to be used for initializing the particle values. If a cell is marked as almost empty, i.e. the number of particles is lower than a preset lower bound for the number of particles per cell, the particle deficit is complemented by creating new particles. The locations for the new particles in their hosting cell are determined using a random number generator. To initialize the other
particle quantities, two options are at the user’s convenience: the particle value is either initialized based on a point interpolation from the underlying mesh field, or the particle value is assigned based on rounding-off the interpolated field value to a lower or upper boundary, i.e.

$$\psi_p = \begin{cases} 
\psi_{\text{min}} & \text{if } \psi_h(x_p) \leq \frac{\psi_{\text{min}} + \psi_{\text{max}}}{2}, \\
\psi_{\text{max}} & \text{if } \psi_h(x_p) > \frac{\psi_{\text{min}} + \psi_{\text{max}}}{2}.
\end{cases} \tag{8}$$

This feature is particularly useful when the particles carry binary fields, such as the density in two-fluid simulations.

The minimal example below demonstrates the LEOPART implementation using the two options for particle insertion/deletion. Results are depicted in Fig. 3.

**Python code**

```python
from dolfin import (UnitSquareMesh, FunctionSpace, Function, Expression, Point, Constant)
from leopart import (particles, RandomRectangle, AddDelete, assign_particle_values)

msh = UnitSquareMesh(2, 2)
Psi = FunctionSpace(msh, "DG", 1)
psi = Function(Psi)

psi_expression = Expression("x[0]", degree=1)
psi.assign(psi_expression)

(p_min, p_max) = (4, 8)
(psi_min, psi_max) = (0., 1.)

# Initialize particles
xp = RandomRectangle(Point(0., 0.), Point(1., 1.)).generate([1, 1])
psip_smooth = assign_particle_values(xp, psi_expression)
psip_binary = assign_particle_values(xp, psi_expression)

p = particles(xp, [psip_smooth, psip_binary], msh)

# Slot 2, psip_binary initialized using binary values
AD = AddDelete(p, p_min, p_max, [psi, psi], [2], [psi_min, psi_max])
# Sweep over mesh to delete/insert particles
AD.do_sweep()

(xp_n, psip_smooth, psip_binary) = (p.return_property(msh, 0),
                                    p.return_property(msh, 1),
                                    p.return_property(msh, 2))
```

The **AddDelete** class can also be used for keeping control over the maximum number of particles per cell by specifying the variable `p_max` in the above presented code. If a cell in the `do_sweep` method is marked to contain more particles than prescribed, the surplus of, say, `m` particles is removed by deleting `m` particles.
with the shortest distance to another particle in that cell. This procedure ensures that particles are removed evenly from the cell interior.

As a final remark: an upwind initialization of the particle value, i.e. initializing the particle value near open boundaries based on the value at the (inflow) boundary facet, is expected to be a useful feature not yet included in LEOPART.

3.3.3. Closed boundaries

When a particle hits a closed boundary during the time step, the particle is reflected by setting the particle velocity to the reflected value, i.e.

\[ \mathbf{a}_h(x_p, t) = \mathbf{a}_h(x_p, t) - 2 (\mathbf{a}_h(x_p, t) \cdot \mathbf{n}) \mathbf{n}, \]

in which \( \mathbf{n} \) the outwardly directed unit normal vector to a boundary facet.

4. Particle-mesh interaction

Obtaining mesh fields from the scattered particle data and updating the particle values from a known mesh field is essential to active tracer problems. These particle-mesh interaction steps go by various names in the literature such as: ‘gather-scatter’ steps \[25, 29\], ‘forward interpolation - backward estimation’ \[30\] or ‘particle weighting’ \[31\].

In line with our earlier work on particle-mesh schemes \[17, 18, 19\], the data transfer operators are consistently coined ‘particle-mesh projection’ for the data transfer from the set of scattered particles to the mesh, whereas the opposite route is indicated by ‘mesh-particle projection’. This convention reflects that the data transfer operators are perceived as projections between different spaces. More precisely, information needs to be projected from a particle space onto a mesh space and vice versa. Adopting this point of view, it
readily follows that the data transfer operations are auxiliary steps, which should not deteriorate accuracy, violate consistency, or compromise on conservation.

To comply with these requirements LEOPART adopts a variational approach to formulate the particle-mesh and the mesh-particle projections. An $\ell^2$ objective function is the starting point for deriving the mutual particle-mesh interactions. For a scalar-valued mesh field $\psi_h$ and a scalar-valued particle field $\psi_p$, this objective function reads

$$\min J := \frac{1}{2} \sum_{p \in S_t} (\psi_h(x_p(t), t) - \psi_p(t))^2,$$  \hspace{1cm} (10)

where it remains to specify the minimizer, other than to say that either $\psi_h$ or $\psi_p$ are used to fit this purpose. The implementation of the projection strategies which can be formulated based on Eq. (10) are further highlighted for a scalar quantity $\psi$ in the remainder of this section, and the projections for a vector-valued quantity follow the same path. More specifically, Section 4.1 discusses the various particle-mesh projections available in LEOPART, and in Section 4.2 the available mesh-particle projections are discussed. The notation $T := \{K\}$ is used throughout to indicate the set of disjoint cells $K$ that constitutes a meshing of the domain $\Omega$, and each cell $K$ has a boundary $\partial K$.

4.1. Particle-mesh projections

Common to the available particle-mesh projections in LEOPART is the minimization of the objective function Eq. (10) with respect to an unknown mesh field $\psi_h$ given a known particle field $\psi_p$. For this, the function space in which $\psi_h$ is approximated must be defined. To this end, LEOPART conveniently exploits existing FENICS tools for defining arbitrary order polynomial function spaces. For reasons that become clear shortly, LEOPART is tailored for projecting the particle data onto discontinuous function spaces at the mesh. In the scalar-valued setting these function spaces are defined by

$$W_h := \{w_h \in L^2(T), \ w_h|_K \in P_k(K) \ \forall \ K \in T\},$$  \hspace{1cm} (11)

where $T$ is the partitioning of the domain $\Omega$ into a set of cells $K$, and $P_k(K)$ denotes the space spanned by Lagrange polynomials on $K$, where the subscript $k \geq 0$ indicates the polynomial order.

4.1.1. $\ell^2$-projection

With these definitions, the most elementary particle-mesh projection is found by minimizing Eq. (10) for $\psi_h \in W_h$, which results in the $\ell^2$-projection: given the particle values $\psi^n_p \in \Psi$, and particle positions $x_p \in X$, find $\psi_h \in W_h$ such that

$$\sum_{p \in S_t} (\psi_h(x_p(t), t) - \psi_p(t)) w_h(x_p(t)) = 0 \hspace{1cm} \forall \ w_h \in W_h.$$  \hspace{1cm} (12)
Given the definition for $W_h$ in Eq. (11), $\psi_h, w_h \in W_h$ are discontinuous across cell boundaries. Hence, Eq. (12) is solved in a cellwise fashion, i.e.

$$
\sum_K \sum_{p \in S_K} (\psi_h(x_p(t), t) - \psi_p(t)) w_h(x_p(t)) = 0 \quad \forall w_h \in W_h, \quad (13)
$$

requiring the inversion of small, local matrices only, thus being amenable to an efficient, parallel implementation.

The particle-mesh projection via the $\ell^2$ projection is implemented in LEoPART in the `l2projection` class, which is instantiated as

```cpp
l2projection(particles& P, FunctionSpace& V, const std::size_t idx);
```

in which the integer index `idx` indicates which particle property is projected. Projection onto a discontinuous space as in Eq. (13), is done with the `project` method, which on the Python side can be invoked as

```python
# FEniCS discrete function space
k = 2
Wh = FunctionSpace(msh, "DG", k)
psi = Function(Wh)
...
# LEoPart particles and l2 projection
p = particles(xp, [psi_p], msh)
lstsq_rho = l2projection(p, Wh, 1)
lstsq_rho.project(psi)
```

LEoPART also allows projection of particle data onto a continuous Galerkin space - which leads to a global system for Eq. (12) - by means of the `project_cg` method

```python
Wh_CG = FunctionSpace(msh, "CG", k)
psi_CG = Function(Wh_CG)
...
lstsq_rho = l2projection(p, Wh_CG, 1)
lstsq_rho.project(psi_CG)
```

4.1.2. Bounded $\ell^2$-projection

The minimization problem, Eq. (12), can be interpreted as a quadratic programming problem. This class of problems has been thoroughly analyzed in literature, and well-known techniques exist to extend these problems with equality, inequality, and box constraints, see e.g. [32] and references. In the context of the particle-mesh projection, imposing box constraints of the form

$$
 l \leq \psi_h \leq u, \quad (14)
$$
can be particularly useful to ensure that the mesh field is bounded by \([l, u]\).

In LEOPART, the box-constrained \(\ell^2\) projection is implemented via a specialization of the \texttt{project} method, which can be invoked as

\[
\begin{align*}
(lb, ub) &= (0., 1.) \\
\text{ltsq\_rho} &= \text{l2projection}(p, Wh, 1) \\
\text{ltsq\_rho\_project}(\psi, lb, ub)
\end{align*}
\]

with \(lb\) and \(ub\) the user-specified lower- and upper bound, respectively. At the backend, LEOPART uses \texttt{QuadProg++} for solving the box-constrained optimization problem. The bounded \(\ell^2\)-projection is only available when projecting onto discontinuous function spaces.

### 4.1.3. PDE-constrained particle-mesh interaction

The motivation for introducing Lagrangian particles - particularly when used as active tracers - is to conveniently accommodate advection. The particle-mesh projections presented in the preceding two sections, however, do not possess conservation properties. That is, initializing a particle quantity from an initial mesh field, advecting the particles, and subsequently reconstructing a mesh field from the updated particle positions with the (box-constrained) \(\ell^2\)-projection, results in a reconstructed mesh field with different integral properties. One way to conserve the mesh properties over the sequence of particle steps, is to keep track of the integral quantities on the mesh. This is accomplished by constraining the objective function for the particle-mesh projection to obtain fields \(\psi_h\) that satisfy a hyperbolic conservation law. The resulting PDE-constrained particle-mesh projection, developed in [18], possesses local (i.e. cellwise) and global conservation properties, and essentially involves solving the minimization problem: given \(\psi_p\), find \(\psi_h \in W_h\)

\[
\min_{\psi_h \in W_h} J = \sum_p \frac{1}{2} \left( \psi_h(x_p) - \psi_p \right)^2
\]

such that:

\[
\begin{align*}
\frac{\partial \psi_h}{\partial t} + \nabla \cdot (a \psi_h) &= 0 \\
&+ \text{BC's}
\end{align*}
\]

is satisfied in a weak sense. For brevity, only periodic boundaries or boundaries with vanishing normal velocity (i.e. \(a \cdot n = 0\)) are considered in this paper. For a more elaborate discussion on other boundary conditions in Eq. (15), reference is made to [18].

By casting the strong form of the constraint into a weak form by multiplying Eq. (15b) with a Lagrange multiplier field \(\lambda_h \in T_h\) - with \(T_h\) defined in Appendix \(A\), Eq. (A.2) - and after applying integration by

\[\text{https://github.com/liuq/QuadProgpp}\]
parts, the PDE-constrained optimization problem amounts to finding the stationary points of the Lagrangian functional

\[ \mathcal{L}(\psi_h, \bar{\psi}_h, \lambda_h) = \sum_p \frac{1}{2} (\psi_h(x_p(t), t) - \psi_p(t))^2 + \sum_K \int_{\partial K} \frac{1}{2} \beta (\psi_h - \bar{\psi}_h)^2 \, d\Gamma + \sum_K \int_{\partial K} \frac{1}{2} \|\nabla \psi_h\|^2 \, d\Omega + \int_{\Omega} \frac{\partial \psi_h}{\partial t} \lambda_h \, d\Omega - \sum_K \int_K a \psi_h \cdot \nabla \lambda_h \, d\Omega + \sum_K \oint_{\partial K} a \cdot n \bar{\psi}_h \lambda_h \, d\Gamma, \quad (16) \]

in which the first term at the right-hand side is similar to the objective function in Eq. (15a). The second line in Eq. (16) is a weak statement of the constraint equation, Eq. (15b). Furthermore, \( \beta > 0 \) is a small penalty parameter introduced to establish a coupling between \( \psi_h \), and the control variable \( \bar{\psi}_h \), where this control variable is defined on the facets of the cell via the trace space \( \bar{W}_h \) from Eq. (A.3), analogous to the flux variable in HDG methods, see, e.g., \([33, 34, 35, 22]\). Finally, \( \zeta \) is a parameter which penalizes gradients, where this parameter is set to zero for smooth problems, and is only invoked when steep gradients in the mesh solution are to be expected, see Section 5.2.

A more in-depth interpretation of Eq. (16) as well as a detailed derivation and analysis of the optimality system, which results after taking variations with respect to the three unknowns \( (\psi_h, \lambda_h, \bar{\psi}_h) \in (W_h, T_h, \bar{W}_h) \), can be found in \([18]\). Appendix A provides a summary of the resulting variational forms in the fully-discrete setting, resulting in a \( 3 \times 3 \) block system at the element level, see Eq. (17). Using notations similar to Eq. (17), LEOPART implements the PDE-constrained particle-mesh projection via the PDEStaticCondensation class

**C++ code**

```cpp
PDEStaticCondensation(const Mesh& mesh, particles& P, const Form& N, const Form& G, const Form& L, const Form& H, const Form& B, const Form& Q, const Form& R, const Form& S, std::vector<std::shared_ptr<const DirichletBC>> bcs, const std::size_t idx_pproperty);
```

To instantiate this class, the weak forms as provided in FormsPDEMap.py are used, and a Python implementation may read

**Python code**

```python
# FEniCS
W = FunctionSpace(mesh, "DG", k)
T = FunctionSpace(mesh, "DG", 0)
Wbar = FunctionSpace(mesh, "DGT", k)

psi_h, psi0_h = Function(W), Function(W)
lambda_h = Function(T)
psibar_h = Function(Wbar)
```

16
bc = DirichletBC(Wbar, Constant(0.), "on_boundary")
...

# LEOPart
FuncSpace_adv = {'FuncSpace_local': W, 'FuncSpace_lambda': T, 'FuncSpace_bar': Wbar}
forms_pde = FormsPDEMap(mesh, FuncSpace_adv).forms_theta_linear(psi0_h, a_advection, dt, Constant(1.0))
pde_projection = PDEStaticCondensation(mesh, p, forms_pde['N_a'], forms_pde['G_a'], forms_pde['L_a'], forms_pde['H_a'], forms_pde['B_a'], forms_pde['Q_a'], forms_pde['R_a'], forms_pde['S_a'], [bc], property_idx)

Assembly of the matrices and vectors is done via the `assemble` method

```python
pde_projection.assemble()
```

which serves a two-fold purpose: first of all it computes the element contributions for each cell $K$ in the $3 \times 3$ block matrix

$$
\begin{bmatrix}
M_p + N & G(\theta) & L \\
G(\theta)^\top & 0 & H \\
L^\top & H^\top & B
\end{bmatrix}
\begin{bmatrix}
\psi^{n+1} \\
\lambda^{n+1} \\
\varphi^{n+1}
\end{bmatrix} =
\begin{bmatrix}
X_p \psi^n_p \\
G(1 - \theta)^\top \psi^n \\
0
\end{bmatrix},
$$

(17)

where the different contributions readily follow from Appendix A, Eq. (A.4).

Secondly, the `assemble` method assembles the local contributions into a global matrix-vector system. Since $\psi_h$ and $\lambda_h$ are local to a cell, the resulting global system of equations is expressed in terms of the flux variable $\bar{\psi}_h$ only. That is, the `assemble` method assembles the global system as follows

$$\bigwedge_K \begin{bmatrix}
\begin{bmatrix}
L & H
\end{bmatrix}^\top & \begin{bmatrix}
M_p + N & G(\theta)
\end{bmatrix}^{-1} & \begin{bmatrix}
L \\
H
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\psi^{n+1}_h \\
\lambda^{n+1}_h \\
\varphi^{n+1}_h
\end{bmatrix} =
- \bigwedge_K \begin{bmatrix}
L & H
\end{bmatrix}^\top \begin{bmatrix}
M_p + N & G(\theta) \\
G(\theta)^\top & 0
\end{bmatrix}^{-1} \begin{bmatrix}
X_p \psi^n_p \\
G(1 - \theta)^\top \psi^*,n
\end{bmatrix},
$$

(18)

in which the wedge $\bigwedge$ denotes assembly of the cell contributions into the global matrix, where this requires the inversion of a small saddle-point problem for each cell $K$ independently, so that the assembly procedure is amenable to a fast parallel implementation.

The method `solve_problem`

```python
pde_projection.solve_problem(psibar_h, psi_h, solver="none", preconditioner="default")
```

solves the resulting global system, Eq. (18), for $\psi^{n+1}_h$. The `solver` and `preconditioner` can be specified by the user and defaults to the MUltifrontal Massively Parallel sparse direct Solver (MUMPS).
to solving the global problem, the solve problem method also applies the back substitution

$$\begin{bmatrix} \psi^{n+1} \\ \lambda^{n+1} \end{bmatrix} = \begin{bmatrix} M_p + N & G(\theta) \\ G^T(\theta) & 0 \end{bmatrix}^{-1} \begin{bmatrix} X_p \psi^{n+1}_p \\ G(1 - \theta)^T \psi^{n+1} \end{bmatrix} - \begin{bmatrix} L \\ H \end{bmatrix} \psi^{n+1} \right), \tag{19}$$

for obtaining the local unknowns $\psi^{n+1}$ and (optionally) the Lagrange multiplier unknowns $\lambda^{n+1}$.

4.2. Mesh-particle projection

The mesh-particle projections, for updating particle properties from a given mesh field, also take the objective functional Eq. (10) as their starting point. Contrary to the particle-mesh projections, the particle-field is the unknown, so that the objective function needs minimization with respect to the particle field $\psi_p$, for the projection of a scalar-valued quantity. Performing the minimization results in: given $\psi_h \in W_h$, find $\psi_p \in \Psi_t$ such that

$$\sum_{p \in S_t} (\psi_h(x_p(t), t) - \psi_p(t)) \delta \psi_p = 0 \quad \forall \ p \in S_t. \tag{20}$$

Since this equation must hold for arbitrary variations $\delta \psi$, the particularly simple result for the mesh-particle projection becomes

$$\psi_p(t) = \psi_h(x_p(t), t) \quad \forall \ p \in S_t, \tag{21}$$

i.e. particles values are obtained via interpolation of the mesh field. Interpolating a mesh field to particles is done in LEOPART via the interpolate method in the particle class, i.e.

```python
p = particles(xp, [psi_p], mesh)
p.interpolate(psi_h, 1)
```

An interpolation overwrites the particle quantities with the interpolated mesh values. However, one of the assets of combining a high resolution particle field with a comparatively low-resolution mesh field is that the particle field may provide sub-grid information to the mesh \cite{36, 37, 28}. In order to take advantage of this, the particles need to have a certain degree of independence from the mesh field. This is achieved by updating the particle quantities by projecting the change in the mesh field rather than overwriting particle quantities. For a scalar valued quantity, this incremental update reads

$$\dot{\psi}_p = \dot{\psi}_h(x_p) \quad \forall \ p \in S_t, \tag{22}$$

in which $\dot{\psi}_h \in W_h$ the time derivative of the mesh field.

A fully-discrete implementation of Eq. (22) is implemented in LEOPART using the $\theta$ method for the time discretization:

$$\psi^{n+1}_p = \psi^n_p + \Delta t \left( (1 - \theta) \dot{\psi}_h^n (x_p^n) + \theta \dot{\psi}_h^{n+1} (x_p^{n+1}) \right) \quad \forall \ p \in S_t, \tag{23}$$
where $\Delta t$ the time step, $1/2 \leq \theta \leq 1$, and $\psi^n_h \in W_h$ is defined as $\dot{\psi}_h^n = (\psi^n_h - \psi^{n-1}_h)/\Delta t$. Eq. (23) is available in LEOPART via the increment method in the particles class, and can be used as

```python
# Particle
p = particles(xp, [psi_p, dpsi_p_dt, msh)

# Incremental update with theta method
theta = 0.5
...
step = 2
p.increment(psih_new, psih_old, [1, 2], theta, step)
```

Two closing remarks are made in view of this incremental update:

- For step 1, $\theta = 1$ since $\dot{\psi}^0_h$ is usually not defined.
- The increment from the old time level, i.e. $\dot{\psi}_p^n (x^n_p)$ is stored at the particle level between consecutive time steps, for efficiency reasons. This requires an additional slot on the particles, i.e. $dpsi_pdt$. The integer array in the increment call indicates which particle slots are used for the incremental update, i.e. $p.increment(psih_new, psih_old, [1, 2], theta, step)$.

5. Example applications

This section demonstrates the performance of LEOPART in terms of accuracy, conservation and computational run time for a number of advection-dominated problems. On a per-time-step basis, the particle-mesh approach typically comprises the following sequence of steps:

1. Lagrangian advection of the particles, as outlined in Section 3.2.
2. particle-mesh projection to project the scattered particle data onto an Eulerian mesh field using either the $\ell^2$-projection, discussed in Section 4.1.1, the bounded constrained $\ell^2$-projection from Section 4.1.2, or the PDE-constrained projection, Section 4.1.3.
3. (optional) solve the physical problem - e.g. a diffusion or Stokes problem - at the mesh, using the reconstructed mesh field as a source term or as an intermediate solution to the discrete equations.
4. (optional) update the particles given the solution at the mesh, using the mesh-particle interaction tools from Section 4.2.

Step 1 and 2 are sufficient to solve an advection problem at the particles and to test the reconstruction of mesh fields from the moving particles. The sequence of steps 1-4 can be used for active tracer modelling or a particle-mesh operator splitting for, e.g., the advection-diffusion equation or the incompressible Navier-Stokes equations, see also Maljaars [17, 18]. For all the examples presented below, reference is made to the corresponding computer code in the LEOPART repository on Bitbucket.
5.1. Translation of a periodic pulse

As a straightforward, yet illustrative example, the translation of the sinusoidal profile

\[ \psi(x, 0) = \sin 2\pi x \sin 2\pi y \] (24)

on the bi-periodic unit square is considered, in analogy to LeVeque [38]. Owing to its simplicity, this test allows to assess the accuracy and the convergence properties of the \( \ell^2 \)- and the PDE-constrained particle-mesh projection. Furthermore, it is used to illustrate the performance of the scheme by means of a strong-scaling study. Test results can be reproduced by running \texttt{SineHump_convergence.py} for the convergence study, and \texttt{SineHump_hires.py} for the scaling study.

In this example, the advective velocity field \( \mathbf{a} = [1, 1]^T \) is used, so that at \( t = 1 \) the initial data should be recovered. To investigate convergence, a coarse mesh, consisting of \( 11 \times 11 \times 2 \) regular triangular cells is refined 5 times where the finest mesh contains \( 176 \times 176 \times 2 \) cells. Different polynomial orders \( k = 1, 2, 3 \) are used for the discontinuous function space \( W_h \), Eq. (11), onto which the particle data is projected. Particles are seeded in a regular lattice on the mesh, such that each cell contains approximately 15 particles, independent of the mesh resolution, see Fig. 4 as an example. An Euler scheme suffices for exact particle advection, and the time step corresponds to a CFL-number of approximately 1. Furthermore, in the PDE-constrained particle-mesh projection, the \( \beta \)-parameter is set to 1e-6, and \( \zeta \) is set to 0. All computations use a direct sparse solver (\texttt{SuperLU}) to solve the global system of equations. Also, note that for this advection problem, the scalar valued property \( \psi_p \), attached to each particle \( p \), needs no updating and stays constant throughout the computation.
Table 1: Translating pulse: overview of model runs with the associated $L^2$-error $\| \psi - \psi_h \|$, the convergence rate and the local mass conservation error $\epsilon_{\Delta \psi_h}$ at time $t = 1$.

| Projection | $\Delta t$ | Cells | Parts. | $(k, l) = (1, 0)$ | $(k, l) = (2, 0)$ | $(k, l) = (3, 0)$ |
|------------|------------|-------|--------|-----------------|-----------------|-----------------|
| $\ell^2$   | 1e-1       | 242   | 3984   | 3.3e-2          | 1.7e-3          | 9.4e-5          |
|            | 5e-2       | 968   | 14542  | 8.3e-3          | 2.1e-4          | 5.9e-6          |
|            | 2.5e-2     | 3872  | 57663  | 2.1e-3          | 2.7e-5          | 3.7e-7          |
| PDE        | 1.25e-2    | 15488 | 230428 | 5.2e-4          | 3.3e-6          | 2.3e-8          |
|            | 6.25e-2    | 61952 | 921837 | 1.3e-4          | 4.1e-7          | 1.4e-9          |

The accuracy of the method is assessed at $t = 1$, using both the $\ell^2$-particle-mesh projection from Section 4.1.1 and the PDE-constrained projection from Section 4.1.3 upon refining the mesh and the time step, see Table 1. Optimal convergence rates of order $k + 1$ are observed for both projections strategies, thus highlighting the accuracy of the particle-mesh projections in conjunction with particle advection.

![Figure 5: Translating pulse: mass error over time for different particle-mesh projections.](image)

As reported in Table 1, the error levels for the $\ell^2$- and the PDE-constrained particle-mesh are similar. The difference between these projections becomes clear, however, by investigating the mass error

$$
\epsilon_{\Delta \psi_h} = \left| \int_\Omega (\psi_h(x, T) - \psi_h(x, 0)) \, d\Omega \right|,
$$

(25)
which is plotted as a function of time for the $\ell^2$-projection in Fig. 5a, and for the PDE-constrained projection in Fig. 5b. Evident from these figures is that the $\ell^2$-projection yields a nonzero mass error, whereas for the PDE-constrained projection the mass error is zero to machine precision.

The trade-off between the non-conservative $\ell^2$-projection and the conservative PDE-constrained is elucidated by investigating the computational times. Wallclock times for the high-resolution case - polynomial order $k = 3$ with 61,952 cells, 921,837 particles and 160 time steps - run on different number of Intel Xeon CPU E5-2690 v4 processors are presented in Fig. 6. Solving the global system for the PDE-constrained projection using a direct solver is computationally much more demanding compared to the (local) $\ell^2$-projection. This illustrates the need for an efficient iterative solver for the PDE-constrained projection step.

![Figure 6: Translating sinusoidal hump: mass error over time for different particle-mesh interactions.](image)

![Figure 6: Translating sinusoidal hump: mass error over time for different particle-mesh interactions.](image)

Table 2: Translating sinusoidal hump: speed-up of the different model parts in parallel computations benchmarked against 1 processor run.

|                  | $\ell^2$-projection | PDE-constrained projection |
|------------------|---------------------|----------------------------|
|                  | 1 2 4 8 16 24       | 1 2 4 8 16 24              |
| Advect particles | 1 2.1 4.1 8.0 15.1 22.4 | 1 1.9 3.7 7.7 14.8 21.9    |
| Assembling       | - - - - - -         | 1 1.9 3.8 7.4 15.4 23.2    |
| Solve projection | 1 2.1 4.3 8.2 16.1 25.1 | 1 1.6 2.4 3.6 4.8 5.8      |
| Total            | 1 2.1 4.2 8.1 15.6 23.8 | 1 1.7 2.7 4.2 5.8 7.2      |

The scaling of the different components is further investigated in Table 2 summarizing the speed-up for the different tests relative to the run on one processor. The particle advection and the assembly step - with the latter only relevant for the PDE-constrained projection - exhibit excellent scaling properties, which is
explained by the locality of these operations, i.e. these steps are performed cellwise. This also holds true for the $\ell^2$-projection, which amounts to a cellwise projection of the particle properties onto a discontinuous function space, see Section 4.1.1. Clearly, the direct sparse solver for the PDE-constrained projection does not possess optimal scaling properties, which thus appears the limiting factor for the scalability of this step.

5.2. Slotted disk

Combining particle-based and mesh-based techniques appears particularly promising for applications in which sharp flow features are to be preserved. The solid body rotation of a slotted disk after Zalesak [39] is a prototypical example of such problems, and often serves as a benchmark for interface tracking schemes, see [40, 41], among many others. We now use this test to demonstrate the various tools that LEOPart offers for tracking sharp discontinuities in material properties, such as a density jump in immiscible multi-fluid flows.

The problem set-up is as follows. A disk with radius 0.2 - from which a slot with a width of 0.1 and depth 0.2 is cut out - is initially centered at $(x, y) = (-0.15, 0)$ on the domain of interest $\Omega := \{(x, y)|x^2+y^2 \leq 0.5\}$. This domain is triangulated into 14,464 cells on which 438,495 particles are seeded. The advective velocity field is given by

$$\mathbf{a} = \pi (-y, x)^\top.$$  

(26)

The time step is set to 0.02, so that one full rotation is performed in 100 steps. The three-stage Runge-Kutta scheme, available via the $\text{advect}\_\text{rk3}$ class, is used for the particle advection.

| Case  | Particle-mesh projection | $k$ | $l$ | $\zeta$ |
|-------|--------------------------|-----|-----|--------|
| Case 1 | Bounded $\ell^2$            | 1   | 0   | -      |
| Case 2 | PDE-constrained           | 1   | 0   | 0      |
| Case 3 | PDE-constrained           | 1   | 0   | 30     |

Different particle-mesh projection strategies that are available in LEOPart are investigated in this example, see Table 3. Note that Case 2 and Case 3 only differ in terms of the $\zeta$ parameter, see Eq. (16). The computer code in $\text{SlottedDisk}\_\text{rotation}\_\text{l2}.py$ reproduces Case 1, the computer code for reproducing Case 2 and Case 3 is found in $\text{SlottedDisk}\_\text{rotation}\_\text{PDE}.py$. The test is run using 8 Intel Xeon CPU E5-2690 v4 processors.

As for the previous example, there is no updating of the scalar valued property attached to a particle from the mesh-solution for this advection problem. Hence, sharp features at the particle level pertain, and
the particle advection part for all the different cases is equal. Fig. 7 shows the particle field at $t = 0$ (initial field), $t = 1$ (half rotation) and after a full rotation at $t = 2$.

Figure 7: Slotted disk: particle field (case independent) at different time instants. Plot shows every 25th particle for clarity, and color values range between 0 (blue) and 1 (red).

Fig. 8 compares the reconstructed mesh fields for the three different cases at time instants $t = 1$ and $t = 2$. For the bounded $\ell^2$-projection (Case 1), the discontinuity in the particle field is captured at the mesh without under- or overshoot, and values stay within the prescribed bounds $0 \leq \psi_h \leq 1$ to machine precision, see Table 4. Another advantage of this approach is that it is fast, and easy to implement in parallel. However, as reported in Table 4 the mass error for the bounded $\ell^2$-projection is non-zero. The latter issue is overcome by using the conservative PDE-constrained particle-mesh projection for the reconstruction of the mesh fields, Case 2 and Case 3. However, for Case 2 - in which $\zeta = 0$ - localized over- and undershoot is observed near the discontinuities. As argued in [18], this artifact is a resolution issue with the mesh resolution being too coarse to capture the sharp discontinuity at the particle level monotonically. By setting $\zeta$ to a value of 30, i.e. approximately equal to the number of particles per cell, this issue is effectively mitigated, see also the minimum and the maximum values for $\psi_h$ over the entire computation reported in Table 4. Table 4 also confirms global mass conservation for Case 2 and Case 3, local conservation properties for similar examples were demonstrated in [18, 19] and not further shown here.

| Cells | Particles | Wallclock time (s) | $\epsilon_{\Delta\psi_2}(t = 2)$ | $\psi_{h,min}(x, t)$ | $\psi_{h,max}(x, t)$ |
|-------|-----------|-------------------|----------------|-----------------|-----------------|
| Case 1 | 14,464 | 438,495 | 150 | 1.4e-5 | -2.6e-16 | 1.00 |
| Case 2 | 14,464 | 438,495 | 182 | 2.0e-15 | -11.2 | 8.04 |
| Case 3 | 14,464 | 438,495 | 180 | 1.3e-15 | -0.01 | 1.02 |

Table 4: Slotted Disk: runtime, area error $\epsilon_{\Delta\psi_2}$ at $t = 2$ and minimum and maximum values for $\psi_h$ for a bounded $\ell^2$-projection (Case 1), or a PDE-constrained projection with $\zeta = 0$ (Case 2) and $\zeta = 20$ (Case 3).
Figure 8: Slotted Disk: reconstructed mesh field $\psi_h$ at different time instants using a particle-mesh projection based on a bounded $\ell^2$-projection (Case 1), or a PDE-constrained projection with $\zeta = 0$ (Case 2) and $\zeta = 30$ (Case 3).
5.3. Lock exchange test

As an example which is geared towards practical multi-fluid applications, the lock-exchange test is considered. Driven by gravity, a dense fluid current moves underneath a lighter fluid, where a thin vertical membrane initially separates the two fluids. Using LEOPart, density tracking and momentum advection is done using Lagrangian particles, and the incompressible, unsteady Stokes equations are discretized on the mesh using the hybridized discontinuous Galerkin (HDG) method from \[22,23\]. The computer code for this test can be found in \texttt{LockExchange.py}.

For this test, the density ratio \(\gamma\) between the two fluids is 0.92. Furthermore, the domain of interest is \(\Omega := [0, 30] \times [-0.5,0.5]\), which is triangulated into \(2,000 \times 80 \times 2 = 320,000\) regular triangular cells. Using the HDG method \[23\] with \(k = 1\) for solving the Stokes equations, the number of dofs in the global system amounts to 1,288,322. A direct sparse matrix solver (\texttt{SuperLU}) is used to solve the Stokes system. The total number of particles amounts to 9,408,000, where this number stays constant throughout the computation. Furthermore, 800 time steps of size \(\Delta t^* = \Delta t \sqrt{g'/H} = 1.25e^{-2}\) are performed, in which \(H = 1\) the channel height, and \(g' = g(1 - \gamma)\) the reduced gravity with \(\gamma = 0.92\) and \(g = 9.81\).

![Figure 9: Lock exchange: density field at the mesh at \(t^* = 10\) using \(\ell^2\) or PDE constrained particle-mesh projections.](image)

Two different particle-mesh projection configurations are considered. In a first configuration, the local \(\ell^2\)-projections are used to project density and momentum data from the particles to the mesh. The density projection is rendered bound preserving by imposing box constraints on the local least squares problem. The density values attached to a particle stay constant throughout the computation, whereas the specific momentum value attached to a particle is updated using Eq. (23) after the Stokes solve.

In the second configuration, the PDE-constrained projection is used for the projection of density and momentum data from the particles to the mesh. This results in a global problem with 964,160 unknowns for the density projection, and 1,928,320 unknowns for the momentum projection. The global systems resulting from the PDE-constrained projections are solved using a GMRES solver in conjunction with an algebraic multigrid preconditioner, where this solver/preconditioner pair is used as a black-box. Furthermore, for the density projection, \(\zeta = 20\) to penalize over- and undershoot in the reconstructed density fields, and as for the other configuration, the specific momentum value attached to a particle is updated using Eq. (24).

Visually, the mesh density fields at \(t^* = 10\) are comparable in terms of the bulk behavior for both
projections, Fig. 9, although differences in the small scale features are observed. No further attempts are made to interpret and value these small scale differences between the local $\ell^2$-projections and the PDE-constrained projections, other than to say that the PDE-constrained approach results in mass- and momentum-conservative fields, whereas this is not so for the $\ell^2$-projection, see Fig. 10.

Timings are reported in Fig. 11 using 32, 64 and 128 CPU cores on the Peta 4 supercomputing facility of the University of Cambridge. Peta 4 contains 768 nodes, equipped with 2 Intel Xeon Skylake 6142 16-core processors each.

Figure 10: Lock exchange: mass- and momentum conservation error as a function of time for two different particle-mesh projection strategies.

Results provide insight into the performance of the different parts, and indicate which parts of the scheme are critical for obtaining higher performance. Clearly, the computational time is dominated by the global solves for the Stokes system, and the PDE-constrained particle-mesh projections, Fig. 11a. The advantage of using iterative solvers for the PDE-projections also becomes clear from this figure. Even though the system for the momentum projection is larger than the system for the Stokes projection, the wallclock time for the momentum projection is considerably smaller and appears to possess better scaling compared to the
Stokes solve. Therefore, implementing the iterative solver for the Stokes solver proposed in [42] is believed to be an important step for improving the performance, and probably indispensable for problems in three spatial dimensions. Noteworthy to mention is that the \( \ell^2 \) particle-mesh projections exhibit excellent scaling properties, on top of their low computational footprint, see Fig. 11b.

![Figure 11: Lock exchange: strong scaling study.](image)

(a) PDE-constrained projections  
(b) \( \ell^2 \)-projections

Table 5: Lock exchange test: speed-up of the different model parts in parallel computations, benchmarked against 32 processors run.

|                         | PDE-projections | \( \ell^2 \) projections |
|-------------------------|----------------|--------------------------|
|                         | # Processors   | # Processors             |
| Particle advection      | 1 1.85 3.87    | 1 2.06 3.95              |
| Density projection      | 1 1.48 1.87    | 1 1.75 3.75              |
| Momentum projection     | 1 1.73 2.25    | 1 1.79 3.99              |
| Stokes solve            | 1 0.93 1.17    | 1 0.99 1.05              |
| Total                   | 1 1.11 1.17    | 1 1.01 1.1               |

5.4. Rayleigh–Taylor instability benchmark

As a last example, the applicability of LEOPART for simulating active particle tracing problems is demonstrated. A well-established benchmark from the geodynamics community is used to fit this purpose, and we consider the Rayleigh-Taylor instability community benchmark proposed by van Keken and co-workers [12]. This benchmark involves the evolution of a geodynamic laminar flow whose initial state is a compositionally light material, situated under a considerably thicker and denser layer. Tools from LEOPART are used to discretise the Stokes system using the HDG method, and Lagrangian particles are used for a diffusion-free
tracking of the chemical composition field. Conservative composition fields at the mesh are reconstructed from the particle representation using the PDE–constrained projection. The reconstructed composition field is subsequently used to compute a source term \( f \) and a composition-dependent viscosity \( \eta \) in the Stokes equations. The code for this numerical experiment can be found in `RayleighTaylorInstability.py`, considering the following problem description:

Let the domain \( \Omega \) be the \([0, L] \times [0, 1]\) rectangle, where \( L = 0.9142 \) is the aspect ratio. \( \psi : \Omega \rightarrow [0, 1] \) is the continuum representation of the chemical composition function, with values 0 and 1 corresponding to the light and dense layer. The source term in the momentum component of Stokes’ equations is given by

\[
f = Rb \psi \hat{g},
\]

where \( Rb = 1 \) is the compositional Rayleigh number and \( \hat{g} = (0, -1)^\top \) is the unit vector acting in the direction of gravity. The viscosity of the Stokes system is dependent on the chemical composition function

\[
\eta = \eta_{\text{bottom}} + \psi (\eta_{\text{top}} - \eta_{\text{bottom}}),
\]

where \( \eta_{\text{top}} \) and \( \eta_{\text{bottom}} \) are the viscosities of the initial heavy top and light bottom layers, respectively. The initial state of \( \psi \) is a small perturbation from the rest state of a light layer of depth \( d_b = 0.2 \),

\[
\psi(x, t = 0) = \begin{cases} 
0 & y < d_b + 0.02 \cos \left( \frac{\pi x}{L} \right), \\
1 & \text{otherwise}. 
\end{cases}
\]

The boundary conditions are set such that \( u = 0 \) on the bottom \((y = 0)\) and top \((y = 1)\) boundaries, and \( u \cdot n = 0 \) on the left \((x = 0)\) and right \((x = L)\) boundaries. Additionally, \( \psi = 0 \) on the bottom boundary \((y = 0)\) and \( \psi = 1 \) on the top boundary \((y = 1)\).

The domain is triangulated into \( 40^2 \times 2, 80^2 \times 2 \) and \( 160^2 \times 2 \) regular simplices. The mesh is then displaced in order to align the cells with the initial viscosity discontinuity described in [29]. Each cell is assigned 25 particles - carrying a composition value \( \psi_p \) - resulting in 80,000, 320,000 and 1,280,000 particles in the meshes, respectively. This number of particles remains constant throughout the simulation. Furthermore, the time step size is chosen based on the Courant–Friedrichs–Levy (CFL) criterion, i.e.

\[
\Delta t_j = C_{\text{CFL}} h_{\text{min}} / \max_{t} |u(t_j)|
\]

where \( t_j \) is the \( j \)th time step, \( h_{\text{min}} \) is the minimum mesh cell diameter and \( C_{\text{CFL}} > 0 \) is the CFL parameter, chosen here to be \( C_{\text{CFL}} = 0.5 \). The HDG method is used with \( k = 1 \) to solve the Stokes system. After computing the solution approximation of the Stokes system, the particles are advected. Given \( \psi_p \), the conservative PDE–constrained projection is used to update the composition field \( \psi_h \) on the mesh and thereby the source term \( f \) and the viscosity \( \eta \). The composition function \( \psi_h \) is represented by the \( k = 1 \) DG–finite element space, and we choose \( \zeta = 25 \) to penalise over– and undershoot of the the reconstructed field.

The benchmark [12] considers three cases, \( \eta_{\text{top}} / \eta_{\text{bottom}} \in \{1, 10, 100\} \). For brevity we document the case with a 100–fold difference in the viscosity layers, namely, \( \eta_{\text{top}} = 1 \) and \( \eta_{\text{bottom}} = 0.01 \). For comparison
with \([12]\), the distribution of the 1 280 000 particles in the \(160^2 \times 2\) mesh at simulation times \(t = 500, 1000\) and 1500 are shown in Fig. [12]. Noteworthy to mention is that the particle distribution remains uniform throughout. This feature owes to HDG discretization of the Stokes system, yielding pointwise div-free velocity fields by which the particles are advected \([17]\).

![Figure 12](image1.png)

(a) \(t = 500\)  
(b) \(t = 1000\)  
(c) \(t = 1500\)

Figure 12: The 1 280 000 particles distributed in the \(160 \times 160\) mesh Rayleigh Taylor instability problem at given simulation times. The dark and light colours represent compositionally dense \(\psi = 1\) and light \(\psi = 0\) regions, respectively. Cf. [12].

![Figure 13](image2.png)

(a) RMS velocity  
(b) Mass conservation error

Figure 13: The computed RMS velocity functional and mass conservation error of the Rayleigh Taylor instability problem. Here \(n\) is the number of cells along one axis of the mesh and \(n_p\) is the number of particles used in the simulation.

A quantitative comparison with literature results from \([12, 43]\) is made by investigating the root mean square (RMS) velocity, \(u_{\text{rms}}\), the mass conservation error, \(\epsilon_{\Delta \psi_{\text{h}}}\), and growth rate, \(\gamma\), where

\[
u_{\text{rms}} = \sqrt{\frac{\int_{\Omega} u \cdot u \, d\Omega}{\lambda}}, \quad \epsilon_{\Delta \psi_{\text{h}}} = \left| \int_{\Omega} (\psi_h(x,t) - \psi_h(x,0)) \, d\Omega \right|, \quad \gamma = \frac{\ln(u_{\text{rms}}(t_1)) - \ln(u_{\text{rms}}(0))}{t_1}, \tag{30}
\]
and \( t_1 \) is the simulation time at the first time step. The computed quantities are shown in Fig. 13, and key results are reported in Table 6. This includes comparison to the marker chain method of \([12]\) and the case with no artificial diffusion in \([43]\), showing good agreement for the growth rate \( \gamma \) and the RMS-velocity.

On top of this, discrete conservation for the composition field \( \psi_h \) can be ensured, Fig. 13b, when using the PDE-constrained particle-mesh projection provided by LEoPART.

| Code          | Grid     | Method                        | \( \gamma \) | \( t_{\text{max } u_{\text{rms}}} \) | max \( u_{\text{rms}} \) |
|---------------|----------|-------------------------------|--------------|-----------------------------------|----------------------|
| van Keken et al. [12] | 100 \times 100 | splines / marker chain      | 0.1024       | 51.12                             | 0.01385              |
|               | 80 \times 80 | \( C1 \)-element / marker chain | 0.1025       | 51.23                             | 0.01448              |
| Vynnytska et al. [43] | 40 \times 40 | Taylor–Hood / DG             | 0.0976       | 57.21                             | 0.01140              |
|               | 80 \times 80 | Taylor–Hood / DG             | 0.1018       | 52.11                             | 0.01444              |
|               | 160 \times 160 | Taylor–Hood / DG           | 0.1039       | 51.55                             | 0.01458              |
| This work     | 40 \times 40 | HDG / PDE-constrained projection | 0.08707     | 54.81                             | 0.01208              |
|               | 80 \times 80 | HDG / PDE-constrained projection | 0.09161     | 50.68                             | 0.01428              |
|               | 160 \times 160 | HDG / PDE-constrained projection | 0.09677     | 50.80                             | 0.01436              |

Table 6: Computed functionals from the Rayleigh Taylor instability simulation and comparison with other works.

6. Conclusion and outlook

This paper introduced LEoPART, a library which integrates a suite of tools for Lagrangian particle advection and different particle-mesh interaction strategies into the open source FE library FEniCS. For efficiently implementing the particle advection, particles are tracked on the (unstructured) polygonal mesh using the convex polyhedron method. As demonstrated in the numerical examples, the particle advection exhibits near optimal performance for distributed-memory parallel runs. Furthermore, several options are available for the projection of particle data onto mesh fields and vice versa. In particular, the PDE-constrained particle-mesh projection allows to track particle quantities on the mesh in an accurate, diffusion-free, and conservative manner.

The code for LEoPART is hosted under an open source license at [https://bitbucket.org/jakob_maljaars/leopart](https://bitbucket.org/jakob_maljaars/leopart). The various numerical tests demonstrate that LEoPART is sufficiently mature to be applicable to a range of practical particle tracing problems, and the community is invited to tailor LEoPART to suit their specific needs. At the same time, LEoPART remains work in progress, and possible directions for further development may include:

- **Iterative solvers:** as demonstrated in the numerical examples, all the components exhibit excellent scaling for distributed-memory parallel runs, except for solving the global systems which arise in the PDE-constrained projection and the incompressible Stokes equations. To improve the performance
for these steps, the implementation of scalable iterative solvers heads our wish list. The optimal
preconditioner presented by [42] for similar systems can serve as a starting point.

- **Implementation of particle advection:** the choice for particle advection schemes is, as yet, limited
to an explicit Euler, or a two- or three-stage Runge-Kutta scheme. LEOPart could benefit from the
implementation of other advection schemes, such as multi-step schemes. In addition, the velocity of the
particles is dictated by a velocity field at the mesh. To further increase the applicability of LEOPart,
the option can be added that particles are (partially) advected with a particle-specific velocity, which,
for instance, is induced by a force exerted on the particle or follows from Brownian motion.

- **Particle output:** as yet, a view of the particles and their properties can be dumped to pickle objects.
Writing the output to HDF5 files would be more efficient, particularly so for large scale computations.

- **dolfin-x support:** the LEOPart functionality works with the latest stable version of FEniCS
(v2019.1.0), but is not yet supported in dolfin-x, the development version of FEniCS.

Acknowledgements

The Netherlands Organisation for Scientific Research (NWO) is acknowledged for supporting JMM
through the JMBC-EM Graduate Programme research grant. CNR is supported by EPSRC Grant EP/N018877/1.
NS is supported by the Carnegie Institution for Science President’s Fellowship. NS further wishes to thank
Peter E. van Keken and Cian R. Wilson for their advice.

Appendix A. PDE-constrained particle-mesh interaction

This appendix presents the discrete optimality system which is obtained by equating the variations of
Eq. (16) with respect to the three unknowns \((\psi_h, \lambda_h, \bar{\psi}_h) \in (W_h, T_h, \bar{W}_h)\) to zero, and performing a \(\theta\) time
integration. A detailed derivation can be found in [18].

To set the stage, let \(T := \{K\}\) be the triangulation of \(\Omega\) into open, non-overlapping cells \(K\), having
outward pointing unit normal vector \(n\) on its boundary \(\partial K\). Adjacent cells \(K_i\) and \(K_j\) \((i \neq j)\) share a
common facet \(F = \partial K_i \cap \partial K_j\). The set of all facets (including the exterior boundary facets \(F = \partial K \cap \partial \Omega\))
is denoted by \(F\). The following scalar finite element spaces are defined on \(T\) and \(F\):

\[
W_h := \{w_h \in L^2(T), w_h|_K \in P_k(K) \quad \forall \ K \in T\}, \tag{A.1}
\]

\[
T_h := \{\tau_h \in L^2(T), \tau_h|_K \in P_l(K) \quad \forall \ K \in T\}, \tag{A.2}
\]

\[
\bar{W}_h := \{\bar{w}_h \in L^2(F), \bar{w}_h|_F = P_h(F) \quad \forall \ F \in F\}, \tag{A.3}
\]

in which \(P(K)\) and \(P(F)\) denote the spaces spanned by Lagrange polynomials on \(K\) and \(F\), respectively,
and \(k \geq 1\) and \(l \geq 0\) indicating the polynomial order. Note that \(W_h\) in Eq. (A.1) is equal to Eq. (11).
Given these function space definitions, the fully-discrete optimality system is obtained after taking variations of the Lagrangian functional Eq. (16) with respect to \((\psi_h, \lambda_h, \bar{\psi}_h) \in (W_h, T_h, \bar{W}_h)\) and using a \(\theta\)-method for the time discretization. The fully-discrete co-state equation in this optimality system reads: given the particle field \(\psi_h^n \in \Psi_t\), the particle positions \(x_{h+1}^n \in X_t\), and the intermediate field \(\psi^{*,n}_h \in W_h\), find \((\psi_n^{n+1}, \lambda_n^{n+1}, \bar{\psi}_n^{n+1}) \in (W_h, T_h, \bar{W}_h)\) such that

\[
\sum_{p \in S_t} \left( \psi_h^{n+1}(x_{h+1}^n) - \psi_h^n(x_p^n) \right) w_h(x_p^n) = - \sum_K \int_{\partial K} \beta \left( \psi_h^{n+1} - \psi_h^n \right) w_h \, d\Gamma + \sum_K \int \zeta \nabla \psi_h^{n+1} \cdot \nabla w_h \, d\Omega \\
+ \int_{\Omega} \frac{w_h}{\Delta t_n} \lambda_h^{n+1} \, d\Omega - \theta \sum_K \int_{\partial K} (a w_h) \cdot \nabla \lambda_h^{n+1} \, d\Omega = 0 \quad \forall \ w_h \in W_h. \tag{A.4a}
\]

Correspondingly, the fully-discrete counterpart of the state equation becomes:

\[
\int_{\Omega} \frac{\psi_h^{n+1} - \psi^{*,n}_h}{\Delta t_n} \tau_h \, d\Omega - \theta \sum_K \int_{\partial K} (a \psi_h^{n+1}) \cdot \nabla \tau_h \, d\Omega + \sum_K \int \alpha \cdot n \, \bar{\psi}_h^{n+1} \tau_h \, d\Gamma \\
= (1 - \theta) \sum_K \int (a \psi^{*,n}_h) \cdot \nabla \tau_h \, d\Omega \quad \forall \ \tau_h \in T_h. \tag{A.4b}
\]

Finally, the fully-discrete optimality condition reads

\[
\sum_K \int_{\partial K} \alpha \cdot n \lambda_h^{n+1} \bar{\psi}_h \, d\Gamma + \sum_K \int \beta \left( \bar{\psi}_h^{n+1} - \psi_h^n \right) \bar{w}_h \, d\Gamma = 0 \quad \forall \ \bar{w}_h \in \bar{W}_h. \tag{A.4c}
\]

Note that the Lagrange multiplier \(\lambda_h\) and the control variable \(\bar{\psi}_h\) are conveniently chosen at time level \(n+1\), which is allowed since these variables are fully-implicit, not requiring differentiation in time. Furthermore, the choice \(l = 0\) for the polynomial order of the Lagrange multiplier field \(\lambda_h\) bears specific advantage in that the terms involving the time-stepping parameter \(\theta\) can be dropped, i.e. for \(l = 0\) the Eq. (A.4) becomes independent of \(\theta\).

Eq. (A.4) can be casted in a 3 × 3 block system, see Eq. (17). This system of equations is solved for the three unknowns \((\psi_h^{n+1}, \lambda_h^{n+1}, \bar{\psi}_h^{n+1}) \in (W_h, T_h, \bar{W}_h)\) via a static condensation procedure, see the discussion in Section 4.1.3.

References

[1] A. C. Bagtzoglou, D. E. Dougherty, A. F. B. Tompson, Application of particle methods to reliable identification of groundwater pollution sources, Water Resour. Manag. 6 (1) (1992) 15–23. \(\text{doi:10.1007/BF00872184}\).

[2] E. J. Delhez, J.-M. Campin, A. C. Hirst, E. Deleersnijder, Toward a general theory of the age in ocean modelling, Ocean Model. 1 (1) (1999) 17–27. \(\text{doi:10.1016/S1463-5003(99)00003-7}\).

[3] E. Deleersnijder, J.-M. Campin, E. J. Delhez, The concept of age in marine modelling: I. Theory and preliminary model results, J. Mar. Syst. 28 (3-4) (2001) 229–267. \(\text{doi:10.1016/S0924-7963(01)00026-4}\).
[4] P. J. Tackley, S. D. King, Testing the tracer ratio method for modeling active compositional fields in mantle convection simulations, Geochemistry, Geophys. Geosystems 4 (4) (2003). doi:10.1029/2001GC000214

[5] N. T. Ouellette, H. Xu, E. Bodenschatz, A quantitative study of three-dimensional Lagrangian particle tracking algorithms, Exp. Fluids 40 (2) (2006) 301–313. doi:10.1007/s00348-005-0088-7

[6] J. Westerweel, G. E. Elsinga, R. J. Adrian, Particle Image Velocimetry for Complex and Turbulent Flows, Annu. Rev. Fluid Mech. 45 (1) (2013) 409–436. doi:10.1146/annurev-fluid-120710-101204

[7] M. Raffel, C. E. Willert, F. Scarano, C. J. Kähler, S. T. Wereley, J. Kompenhans, Particle image velocimetry: a practical guide, Springer, 2018.

[8] E. Hathway, C. Noakes, P. Sleigh, L. Fletcher, CFD simulation of airborne pathogen transport due to human activities, Build. Environ. 46 (12) (2011) 2500–2511. doi:10.1016/J.BUILDENV.2011.06.001

[9] J. Westerweel, G. E. Elsinga, R. J. Adrian, Particle Image Velocimetry for Complex and Turbulent Flows, Annu. Rev. Fluid Mech. 45 (1) (2013) 409–436. doi:10.1146/annurev-fluid-120710-101204

[10] S. Pope, PDF methods for turbulent reactive flows, Prog. Energy Combust. Sci. 11 (2) (1985) 119–192. doi:10.1016/0360-1285(85)90002-4

[11] Y. Zhang, D. Haworth, A general mass consistency algorithm for hybrid particle/finite-volume PDF methods, J. Comput. Phys. 194 (1) (2004) 156–193. doi:10.1016/j.jcp.2003.08.032

[12] P. E. van Keken, S. D. King, H. Schmeling, U. R. Christensen, D. Neumeister, M.-P. Doin, A comparison of methods for the modeling of thermochemical convection, Journal of Geophysical Research: Solid Earth 102 (1997) 22.

[13] U. R. Christensen, A. W. Hofmann, Segregation of subducted oceanic crust in the convecting mantle, Journal of Geophysical Research: Solid Earth 99 (1994) 19. doi:10.1029/93JB03403

[14] J. P. Brandenburg, P. E. van Keken, Deep storage of oceanic crust in a vigorously convecting mantle, Journal of Geophysical Research: Solid Earth 112 (2007) B06403. doi:10.1029/2006JB004813

[15] E. Edwards, R. Bridson, A high-order accurate particle-in-cell method, Int. J. Numer. Methods Eng. 90 (9) (2012) 1073–1088. doi:10.1002/nme.3356

[16] D. M. Kelly, Q. Chen, J. Zang, PICIN: a particle-in-cell solver for incompressible free surface flows with two-way fluid-solid coupling, SIAM J. Sci. Comput. 37 (3) (2015) 403 – 424. doi:10.1137/140976911

[17] J. M. Maljaars, R. J. Labeur, M. Möller, A hybridized discontinuous Galerkin framework for high-order particle-mesh operator splitting of the incompressible Navier-Stokes equations, J. Comput. Phys. 358 (2018) 150–172. doi:10.1016/j.jcp.2017.12.036

[18] J. M. Maljaars, R. J. Labeur, N. Trask, D. Sulsuk, Conservative, high-order particle-mesh scheme with applications to advection-dominated flows, Comput. Methods Appl. Mech. Eng. 348 (2019) 443–465. doi:10.1016/J.CMA.2019.01.028

[19] J. M. Maljaars, R. J. Labeur, N. Trask, D. Sulsuk, Optimization Based Particle-Mesh Algorithm for High-Order and Conservative Scalar Transport, Lecture Notes in Computational Science and Engineering., Springer, 2019, Ch. 23.

[20] J. Maljaars, R. J. Labeur, M. Möller, W. Uijttewaal, Development of a hybrid particle-mesh method for simulating free-surface flows, J. Hydrodyn. Ser. B 29 (3) (2017) 413–422. doi:10.1007/s11800-016-6058-1

[21] A. Logg, K.-A. Mardal, G. N. Wells, Automated Solution of Differential Equations by the Finite Element Method, Vol. 84, Springer Science & Business Media, 2012. doi:10.1007/978-3-642-23099-8

[22] S. Rhebergen, G. N. Wells, A Hybridizable Discontinuous Galerkin Method for the Navier-Stokes Equations with Pointwise Divergence-Free Velocity Field, J. Sci. Comput. (2018) 1–18. doi:10.1007/s10915-018-0671-4

[23] S. Rhebergen, G. N. Wells, An embedded–hybridized discontinuous Galerkin finite element method for the Stokes equations arXiv:1811.09194

[24] M. Muradoglu, A. D. Kayaalp, An auxiliary grid method for computations of multiphase flows in complex geometries, J.
[25] R. Löhere, J. Ambrosiano, A vectorized particle tracer for unstructured grids, J. Comput. Phys. 91 (1) (1990) 22–31. doi:10.1016/0021-9991(90)90002-I

[26] J. Maljaars, A hybrid particle-mesh method for simulating free surface flows (2016). URL http://resolver.tudelft.nl/uuid:d894370d-f6df-4433-8ee0-7692a43e857a

[27] D. Haworth, Progress in probability density function methods for turbulent reacting flows, Prog. Energy Combust. Sci. 36 (2) (2010) 168–250. doi:10.1016/j.pecs.2009.09.003

[28] P. P. Popov, R. McDermott, S. B. Pope, An accurate time advancement algorithm for particle tracking, J. Comput. Phys. 227 (20) (2008) 8792–8806. doi:10.1016/j.jcp.2008.06.021

[29] Z. Chen, Z. Zong, M. B. Liu, H. T. Li, A comparative study of truly incompressible and weakly compressible SPH methods for free surface incompressible flows, Int. J. Numer. Methods Fluids 73 (9) (2013) 625–638.

[30] R. Garg, C. Narayanan, D. Lakehal, S. Subramaniam, Accurate numerical estimation of interphase momentum transfer in Lagrangian-Eulerian simulations of dispersed two-phase flows, Int. J. Multiph. Flow 33 (12) (2007) 1337–1364. doi:10.1016/j.ijmultiphaseflow.2007.06.002

[31] G. Jacobs, J. Hesthaven, High-order nodal discontinuous Galerkin particle-in-cell method on unstructured grids, J. Comput. Phys. 214 (1) (2006) 96–121. doi:10.1016/j.jcp.2005.09.008

[32] D. Goldfarb, A. Idnani, A numerically stable dual method for solving strictly convex quadratic programs, Math. Program. 27 (1) (1983) 1–33. doi:10.1007/BF02591962

[33] R. J. Labeur, G. N. Wells, Energy stable and momentum conserving hybrid finite element method for the incompressible Navier–Stokes equations, SIAM J. Sci. Comput. 34 (2) (2012) 889–913. doi:10.1137/100818583

[34] G. N. Wells, Analysis of an Interface Stabilized Finite Element Method: The Advection-Diffusion-Reaction Equation, SIAM J. Numer. Anal. 49 (1) (2011) 87–109. doi:10.1137/090775464

[35] N. Nguyen, J. Peraire, B. Cockburn, An implicit high-order hybridizable Galerkin method for linear convection–diffusion equations, J. Comput. Phys. 228 (9) (2009) 3232–3254. doi:10.1016/j.jcp.2009.01.030

[36] D. Snider, An Incompressible Three-Dimensional Multiphase Particle-in-Cell Model for Dense Particle Flows, J. Comput. Phys. 170 (2) (2001) 523–549. doi:10.1006/jcph.2001.6747

[37] R. McDermott, S. Pope, The parabolic edge reconstruction method (PERM) for Lagrangian particle advection, J. Comput. Phys. 227 (11) (2008) 5447–5491. doi:10.1016/j.jcp.2008.01.045

[38] R. J. LeVeque, High-Resolution Conservative Algorithms for Advection in Incompressible Flow, SIAM J. Numer. Anal. 33 (2) (1996) 627–665. doi:10.1137/0733033

[39] S. T. Zalesak, Fully multidimensional flux-corrected transport algorithms for fluids, J. Comput. Phys. 31 (3) (1979) 335–362. doi:10.1016/0021-9991(79)90051-2

[40] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A Hybrid Particle Level Set Method for Improved Interface Capturing, J. Comput. Phys. 183 (1) (2002) 83–116. doi:10.1006/jcph.2002.7166

[41] R. Scardovelli, S. Zaleski, Interface reconstruction with least-square fit and split Eulerian-Lagrangian advection, Int. J. Numer. Methods Fluids 41 (3) (2003) 251–274. doi:10.1002/fld.431

[42] S. Rhebergen, G. N. Wells, Preconditioning of a hybridized discontinuous Galerkin finite element method for the Stokes equations, J. Sci. Comput. 77 (3) (2018) 1936–1952. doi:10.1007/s10915-018-0760-4

[43] L. Vynnytska, M. E. Rognes, S. R. Clark, Benchmarking FEniCS for mantle convection simulations, Computers and Geosciences 50 (2013) 95–105. doi:10.1016/j.cageo.2012.08.012