Numerical investigation of linear advection equation with SUPG method

Duan Xu   123010910069

June 12, 2024

1 Introduction

In this paper, the solution of the linear advection equation is investigated with finite element method (FEM).

The advection equation discussed is as follows:

$$\boldsymbol{\beta} \cdot \nabla u = f \tag{1}$$

where $\boldsymbol{\beta}$ is a vector field that describes the advection direction and speed (which may be dependent on the space variables if $\boldsymbol{\beta} = \boldsymbol{\beta}(x)$, f is a source function, and u is the solution. The physical process that this equation describes is that of a given flow field $\boldsymbol{\beta}$, with which another substance is transported, the density or concentration of which is given by u. The equation does not contain diffusion of this second species within its carrier substance, but there are source terms f.

At the inflow, the above equation is augmented by boundary conditions:

$$u = g, \qquad \mathbf{x} \in \partial\Omega_- \tag{2}$$

where $\partial\Omega_-$ is where the substance is transported in, defined by

$$\partial\Omega_- = \{\mathbf{x} : \boldsymbol{\beta} \cdot \mathbf{n}(\mathbf{x}) < 0\}$$

where $\mathbf{n}(\mathbf{x})$ is the normal vector of the field pointing outwards.

No boundary condition should be imposed at the outflow $\partial\Omega_+$.

This equation cannot be solved in a stable way using the standard finite element method. The problem is that solutions to this equation possess insufficient regularity perpendicular to the transport direction: while they are smooth along the streamlines defined by the flow field $\boldsymbol{\beta}$, they may be discontinuous perpendicular to this direction. This is easy to understand: what the equation $\boldsymbol{\beta} \cdot \nabla u = f$ means is in essence that the rate of change of u in direction $\boldsymbol{\beta}$ equals f. But the equation has no implications for the derivatives in the perpendicular direction, and consequently if f is discontinuous at a point on the inflow boundary, then this discontinuity will simply be transported along the streamline of the wind field that starts at this boundary point. These discontinuities lead to numerical instabilities that make a stable solution by a standard continuous finite element discretization impossible.

A standard approach to address this difficulty is the "streamline-upwind Petrov-Galerkin" (SUPG) method, sometimes also called the streamline diffusion method[1].

The computation conducted in this paper is under the framework of `deal.II`[2].

2 Numerical method

This section serves for three aims:

1. Weak form of the linear advection equation is formulated.
2. Developed a simple criterion for mesh refinement.
3. Multiple threads is employed to expedite the assembly process of finite element matrices within a multi-processor machine.

2.1 SUPG

To develop the weak form of the equation, instead of multiplying the equation by a test function v, we multiply $v + \delta\boldsymbol{\beta} \cdot \nabla v$ to the advection equation and integrate in the domain. δ is a parameter that is chosen in the range of the (local) mesh width h.

$$(v + \delta\boldsymbol{\beta} \cdot \nabla v, \boldsymbol{\beta} \cdot \nabla u)_\Omega = (v + \delta\boldsymbol{\beta} \cdot \nabla v, f)_\Omega \tag{3}$$

where $(f, g)_\Omega = \int f \cdot g \mathrm{d}\Omega$

And for the inflow condition, another weight function w is multiplied giving

$$(w, u)_{\partial\Omega_-} = (w, g)_{\partial\Omega_-} \tag{4}$$

For SUPG, w is chosen to be $\boldsymbol{\beta} \cdot \mathbf{n}v$, so

$$(\boldsymbol{\beta} \cdot \mathbf{n}v, u)_{\partial\Omega_-} = (\boldsymbol{\beta} \cdot \mathbf{n}v, g)_{\partial\Omega_-} \tag{5}$$

Combining 3 and 5, the ensuing equation is derived

$$(v + \delta\boldsymbol{\beta} \cdot \nabla v, \boldsymbol{\beta} \cdot \nabla u)_\Omega - (\boldsymbol{\beta} \cdot \mathbf{n}v, u)_{\partial\Omega_-} = (v + \delta\boldsymbol{\beta} \cdot \nabla v, f)_\Omega - (\boldsymbol{\beta} \cdot \mathbf{n}v, g)_{\partial\Omega_-} \tag{6}$$

Write the solution as a combination of base functions

$$u \approx \Sigma_{i=1}^{n} U_i^e \phi_i \tag{7}$$

Since equation 6 hold for any weight functionϕ_i, we can develop the element equation

$$A^K u^K = F^K \tag{8}$$

where

$$\begin{aligned} A_{ij}^K &= (\phi_i + \delta\boldsymbol{\beta} \cdot \nabla\phi_i, \boldsymbol{\beta} \cdot \nabla\phi_j)_\Omega - (\boldsymbol{\beta} \cdot \mathbf{n}\phi_i, \phi_j)_{\partial\Omega_-} \\ F_i^K &= (\phi_i + \delta\boldsymbol{\beta} \cdot \nabla\phi_i, f)_\Omega - (\boldsymbol{\beta} \cdot \mathbf{n}\phi_i, g)_{\partial\Omega_-} \end{aligned} \tag{9}$$

The SUPG method enhance stability by introducing extra diffusion terms. To see this, the modified differential equations based on the weak form developed above is derived to contrast the original linear advection equation. Take out the first term on the left hand from equation 6 and expand

$$(v + \delta\boldsymbol{\beta} \cdot \nabla v, \boldsymbol{\beta} \cdot \nabla u)_\Omega = (v, \boldsymbol{\beta} \cdot \nabla u)_\Omega + (\delta\boldsymbol{\beta} \cdot \nabla v, \boldsymbol{\beta} \cdot \nabla u)_\Omega \tag{10}$$

The second term in 10 can be written as (negelecting boundary term)

$$(\delta\boldsymbol{\beta}\cdot\nabla v,\boldsymbol{\beta}\cdot\nabla u)_\Omega = -(v,\delta(\beta\cdot\nabla)^2u) \tag{11}$$

Hence the weak form is transform to

$$(v,\boldsymbol{\beta}\cdot\nabla u-\delta(\beta\cdot\nabla)^2u) \tag{12}$$

which indicates the modified differential equation

$$\boldsymbol{\beta}\cdot\nabla u-\delta(\beta\cdot\nabla)^2u=f \tag{13}$$

Where the term $-\delta(\beta\cdot\nabla)^2u$ is the added diffution term used to enhance stability.

2.2 Mesh refinement

To adaptively refine mesh in finite element method, it is common to use an error estimator first developed by Kelly et al.[3], which assigns to each cell K the following indicator:

$$\eta_K=\left(\frac{h_K}{24}\int_{\partial K}[\partial_n u_h]^2\mathrm{d}\sigma\right)^{1/2} \tag{14}$$

where $[\partial_n u_h]$ denotes the jump of the normal derivatives across a face $\gamma\subset\partial K$ of the cell K. It can be shown that this error indicator uses a discrete analogue of the second derivatives, weighted by a power of the cell size that is adjusted to the linear elements assumed to be in use here:

$$\eta_K\approx Ch||\nabla^2u||_K \tag{15}$$

which itself is related to the error size in the energy norm.

The problem with this error indicator in the present case is that it assumes that the exact solution possesses second derivatives. Although most problems allow solutions in H^2, solution of advection problem does not belongs to the category. If solutions are only in H^1, then the second derivatives would be singular in some parts (of lower dimension) of the domain and the error indicators would not reduce there under mesh refinement. Thus, the algorithm would continuously refine the cells around these parts, i.e. would refine into points or lines (in 2d).

For advection equation, the solution does not even belongs to H^1, so the error indicator described above is not really applicable. Hence an indicator that is based on a discrete approximation of the gradient is needed. To start with, it is noted that given two cells K,K' of which the centers are connected by the vector $\mathbf{y}_{KK'}$, The directional derivative of the function u can be approximated with the following methodology

$$\frac{\mathbf{y}_{KK'}^T}{|\mathbf{y}_{KK'}|}\nabla u\approx\frac{u(K')-u(K)}{|\mathbf{y}_{KK'}|} \tag{16}$$

where $u(K)$ and $u(K')$ denote u evaluated at the centers of the respective cells. Now multiply the above approximation by $\mathbf{y}_{KK'}/|\mathbf{y}_{KK'}|$ and sum over all neighbors K' of K:

$$\underbrace{\left(\sum_{K'}\frac{\mathbf{y}_{KK'}\mathbf{y}_{KK'}^T}{|\mathbf{y}_{KK'}|^2}\right)}_{=:Y}\nabla u\approx\sum_{K'}\frac{\mathbf{y}_{KK'}}{|\mathbf{y}_{KK'}|}\frac{u(K')-u(K)}{|\mathbf{y}_{KK'}|} \tag{17}$$

If the vectors $\mathbf{y}_{KK'}$ connecting K with its neighbors span the whole space (i.e. roughly: K has neighbors in all directions), then the term in parentheses in the left hand side expression forms a regular matrix, which can be inverted to obtain an approximation of the gradient of u on K:

$$\nabla u \approx Y^{-1}\left(\sum_{K'} \frac{\mathbf{y}_{KK'}}{|\mathbf{y}_{KK'}|}\frac{u(K')-u(K)}{|\mathbf{y}_{KK'}|}\right). \tag{18}$$

Denote the approximation on the right hand side by $\nabla_h u(K)$, and the following quantity is used as refinement criterion:

$$\eta_K = h^{1+d/2}|\nabla_h u_h(K)| \tag{19}$$

2.3 Work streams

In FEM, there are considerable independent jobs: for example, assembling local contributions to the global linear system on each cell of a mesh; evaluating an error estimator on each cell; or postprocessing on each cell computed data for output fall into this class. These cases can be treated using a software design pattern we call WorkStream[4].

In the traditional way, the system matrix is assembled independently and the WorkStream schedules each task, the implementation looks like:

```
template <int dim>
void MyClass<dim>::assemble_system ()
{
    WorkStream::run(dof_handler.begin_active(),
                    dof_handler.end(),
                    *this,
                    &MyClass<dim>::assemble_on_one_cell);
}

template <int dim>
void MyClass<dim>::assemble_on_one_cell (
     const typename DoFHandler<dim>::active_cell_iterator &cell)
{
    FEValues<dim> fe_values;
    FullMatrix<double> cell_matrix;
    Vector<double>     cell_rhs;

    // assemble local contributions
    fe_values.reinit (cell);
    for (unsigned int i=0; i<fe.dofs_per_cell; ++i)
        for (unsigned int j=0; j<fe.dofs_per_cell; ++j)
            for (unsigned int q=0; q<fe_values.n_quadrature_points; ++q)
                cell_matrix(i,j) += ...;
    ...same for cell_rhs...
    // now copy results into global system
    std::vector<unsigned int> dof_indices;
    cell->get_dof_indices (dof_indices);
    for (unsigned int i=0; i<fe.dofs_per_cell; ++i)
        for (unsigned int j=0; j<fe.dofs_per_cell; ++j)
            system_matrix.add (dof_indices[i], dof_indices[j],
                               cell_matrix(i,j));
    ...same for rhs...
}
```

The problem here is that several tasks, each running `MyClass::assemble_on_one_cell`, could potentially try to write into the object `MyClass::system_matrix` at the same time. This could be avoided by explicit synchronisation using a `Threads::Mutex` but this method is not efficient.

As a consequence, the way the WorkStream class is designed is to use two functions: the `MyClass::assemble_on_one_cell` computes the local contributions and stores them, and a second function, say `MyClass::copy_local_to_global`, that copies the results computed on each cell into the global objects. So the implementation is like

```
struct PerTaskData {
    FullMatrix<double>          cell_matrix;
    Vector<double>              cell_rhs;
    std::vector<unsigned int> dof_indices;
}

template <int dim>
void MyClass<dim>::assemble_on_one_cell(
        const typename DoFHandler<dim>::active_cell_iterator &cell,
PerTaskData &data)
{
    FEValues<dim> fe_values;

    data.cell_matrix = 0;
    data.cell_rhs    = 0;

    // assemble local contributions
    fe_values.reinit (cell);
    for (unsigned int i=0; i<fe.dofs_per_cell; ++i)
        for (unsigned int j=0; j<fe.dofs_per_cell; ++j)
            for (unsigned int q=0; q<fe_values.n_quadrature_points; ++q)
                data.cell_matrix(i,j) += ...;
    ...same for cell_rhs...

    cell->get_dof_indices (data.dof_indices);
}
```

```
template <int dim>
void MyClass<dim>::copy_local_to_global (const PerTaskData &data)
{
    for (unsigned int i=0; i<fe.dofs_per_cell; ++i)
        for (unsigned int j=0; j<fe.dofs_per_cell; ++j)
            system_matrix.add(data.dof_indices[i], data.dof_indices[j],
                data.cell_matrix(i,j));
    ...same for rhs...
}

template <int dim>
void MyClass<dim>::assemble_system ()
{
    PerTaskData per_task_data;

    WorkStream::run (dof_handler.begin_active(), dof_handler.end(),
                     *this,
                     &MyClass<dim>::assemble_on_one_cell,
                     &MyClass<dim>::copy_local_to_global,
                     per_task_data);
}
```

There are two keys in the implementation:

1. The `MyClass::copy_local_to_global` never runs more than once in parallel, so the system matrix will never be written by more than one tasks at the same time.

2. The object called `PerTaskData` that is designed to store local infomation will be passed first to one of possibly several instances of `MyClass::assemble_on_one_cell` running in parallel which fills it with the data obtained on a single cell, and then to a sequentially running `MyClass::copy_local_to_global` that copies data into the global object. In practice, these objects are recyled after being used to increase efficiency.

2.4 Other Details on the finite element method

The Lagrange fifth-order rectangular element is used to describe the mesh. In two dimension, there are 36 dofs each element, i.e., 36 nodes, 36 base functions, the dimension of cell matrix is also 36. In three dimension, there are 216 dofs each element.

Method of numerical integration is sixth-order Gauss-Legendre quadrature, i.e., there are 36 quadrature points in two dimension and 216 quadrature points in three dimension each element.

The system matrix is stored as a sparse matrix to save storage and computing time. It is solved with a GMRES (Generalized Minimum RESidual) solver with maximum relative error 10^{-1}.

3 Result discussion

3.1 Test case

For the problem solved in this paper, the domain and functions are given by ($d = 2, 3$ is the space dimension):

$$\begin{aligned}
\Omega =&[-1, 1]^d \\
\boldsymbol{\beta}(\mathbf{x}) =& \begin{pmatrix} 2 \\ 1 + \frac{4}{5}\sin(8\pi\mathbf{x}) \end{pmatrix} \\
s =&0.1 \\
f(\mathbf{x}) =& \begin{cases} \frac{1}{10s^d} & \text{for } |\mathbf{x} - \mathbf{x}_0| < s, \\ 0 & \text{else.} \end{cases} \qquad \mathbf{x}_0 = \begin{pmatrix} -\frac{3}{4} \\ -\frac{3}{4} \end{pmatrix} \\
g(\mathbf{x}) =&\mathrm{e}^{5(1-|\mathbf{x}|^2)}\sin(16\pi|\mathbf{x}|^2) \\
\delta =&0.1h \qquad (h \text{ is element length})
\end{aligned} \tag{20}$$

For $d = 3$, we extend $\boldsymbol{\beta}$ and $\mathbf{x}_0$ by simply duplicating the last of the components shown above one more time.

The input functions have the following meaning:

1. The advection field $\boldsymbol{\beta}$ transports the solution roughly in diagonal direction from lower left to upper right, but with a wiggle structure superimposed.

2. The right hand side adds to the field generated by the inflow boundary conditions a blob in the lower left corner, which is then transported along.

3. The inflow boundary conditions impose a weighted sinusoidal structure that is transported along with the flow field.

3.2 Convergence

The average value defined by

$$\bar{u} = \frac{1}{2^d}\int_\Omega u \mathrm{d}\Omega$$

are used to test convergence.

In 2-dimensional case, the computation is conducted for 6 times, refine grid each time.

Table 1: The number of cells and number of dofs in each cycle in 2-dimensional case

cycle	Number of cells	Number of dofs	$\bar{u}$
1	64	1681	0.162351
2	121	3436	0.164961
3	238	6487	0.166480
4	481	13510	0.168481
5	958	26137	0.168079
6	1906	52832	0.169026

In 3-dimensional case, the computation is conducted for 3 times, refine grid each time.

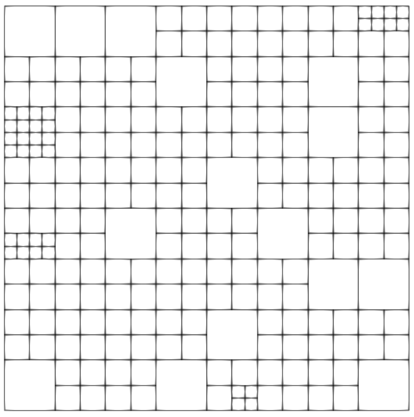
Table 2: The number of cells and number of dofs in each cycle in 3-dimensional case

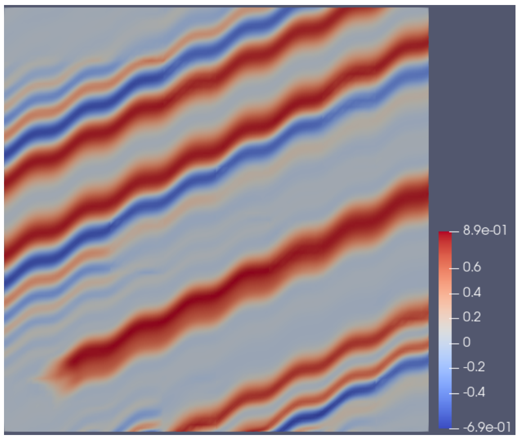
cycle	Number of cells	Number of dofs	$\bar{u}$
1	64	9261	0.0172231
2	197	30443	0.0192791
3	701	104231	0.0199108

The outcomes in both two-dimensional and three-dimensional analysis consistently demonstrate favorable convergence properties.

3.3 Results in 2-dimension

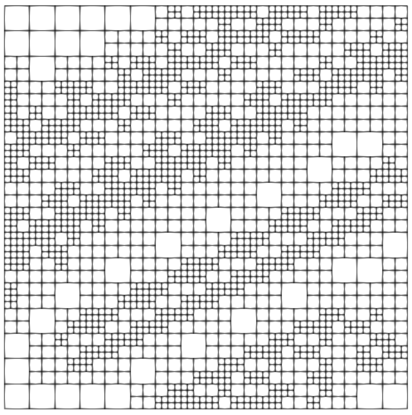
In 2-dimensional case, we show the grid and solution when cycle $= 3, 6$

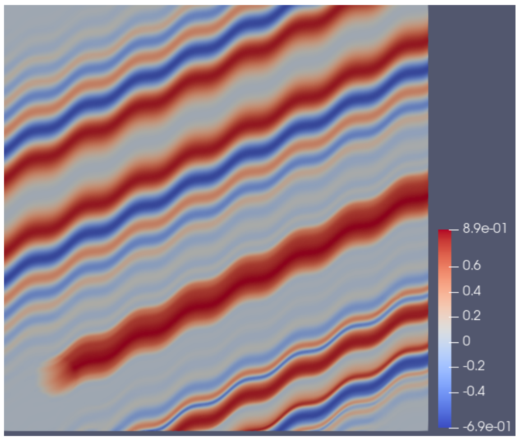

(a) Grid



(b) Solution

Figure 1: The grid and solution when cycle $= 3$


(a) Grid



(b) Solution

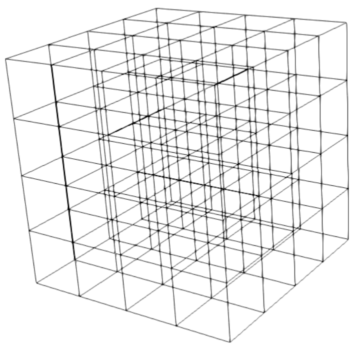
Figure 2: The grid and solution when cycle = 6

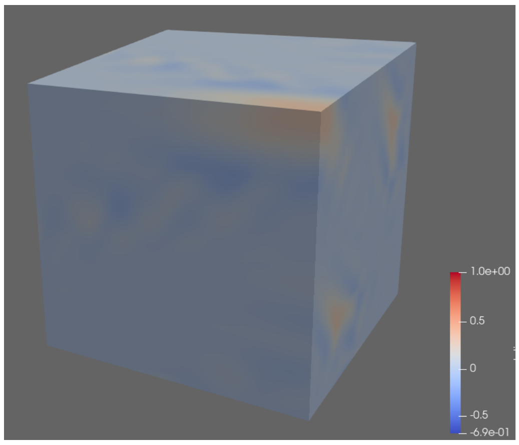
The result shows that:

1. Variable u is transported along the wiggly advection field from the left and lower boundaries to the top right, as expected.
2. The comparison between plots shows that finer grid captures more complex wiggles. The grid shown above is well-adapted to resolve these features.

3.4 Results in 3-dimension

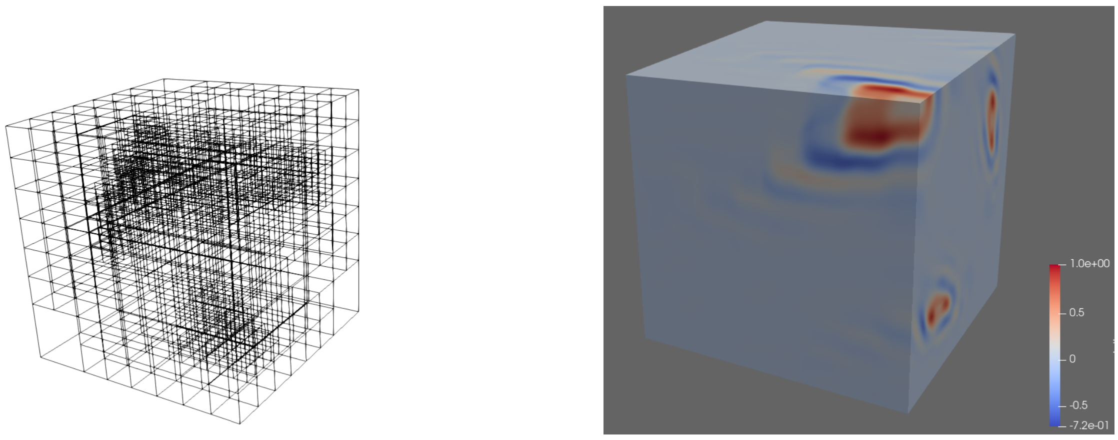
In 3-dimensional case, we show the grid and solution when cycle $= 1, 3$


(a) Grid



(b) Solution

Figure 3: The grid and solution when cycle = 1



(a) Grid (b) Solution

Figure 4: The grid and solution when cycle = 3

The result in three-dimension shows similar features as in two-dimension.

4 Conclusion

1. The SUPG method shows great stability in solving linear advection equation. The intense variation in the direction normal to the advection field is well captured.
2. The proposed mesh refinement criterion performs well in adapting the grid to resolve the wiggles in solution.
3. Parallel computing and multi-threads can be well-utilized in FEM to improve efficiency.

References

[1] H.C. Elman, D.J. Silvester, and A.J. Wathen. Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, New York, 2005.

[2] Daniel Arndt, Wolfgang Bangerth, et al. The `deal.II` Library, Version 9.5. Journal of Numerical Mathematics, 2023, 43(3), 231-246.

[3] D.W. Kelly, J.P. De S. R. Gago, O.C. Zienkiewicz, and I. Babuska. A posteriori error analysis and adaptive processes in the finite element method: Part I–error analysis. International Journal for Numerical Methods in Engineering, 19:1593–1619, 1983.

[4] Bruno Turcksin and Martin Kronbichler and Wolfgang Bangerth. *WorkStream* – a design pattern for multicore-enabled finite element computations. ACM Transactions on Mathematical Software, vol. 43, pp. 2/1-29, 2016.