Series 5. Part I

Numerical methods for PDEs
Last edited: May 18, 2017
Due date: 2017-06-01 at 23:59

Template codes are available on the course's webpage at `https://moodle-app2.let.ethz.ch/course/view.php?id=3089`.

Please keep in mind that part II of this series will be published soon and it will not contain core problems.

Exercise 1 Scalar Conservation Laws in 1D

Consider the general scalar conservation law in one dimension:

$$\partial_t U(x,t) + \partial_x f(U) = 0, \qquad \text{for } (x,t) \in \mathbb{R} \times \mathbb{R}^+. \tag{1}$$

with f the flux function.

1a)

Let us focus on Burgers' equation by considering (1) with flux $f(U) = \frac{U^2}{2}$, i.e.:

$$\partial_t U(x,t) + \partial_x \left(\frac{U(x,t)^2}{2} \right) = 0, \qquad \text{for } (x,t) \in \mathbb{R} \times \mathbb{R}^+. \tag{2}$$

Use the Rankine-Hugoniot condition and/or the Lax-entropy condition to solve (2) with the following initial data:

i)

$$U(x,0) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x > 0 \end{cases}, \qquad \text{for } x \in \mathbb{R}. \tag{3}$$

ii)

$$U(x,0) = \begin{cases} 0 & \text{if } x < 0, \\ -2 & \text{otherwise.} \end{cases} \qquad \text{for } x \in \mathbb{R}. \tag{4}$$

iii)

$$U(x,0) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0. \end{cases} \qquad \text{for } x \in \mathbb{R}. \tag{5}$$

iv)

$$U(x,0) = \begin{cases} 1 & \text{if } 0 < x < 1, \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } x \in \mathbb{R}. \tag{6}$$

1b)

Compute the characteristics for Burgers' equation (2). Plot the characteristics for the following initial data:

$$U(x,0) = \sin(2\pi x), \qquad \text{for } x \in [-1,1]. \tag{7}$$

1c)

(**Core problem**) In the template file scalarconservationlaw.cpp, implement the function

```
void Godunov(int N, double T, const std::function<double(double)>& f,
             const std::function<double(double)>& u0,
             Eigen::VectorXd& u,
             Eigen::VectorXd& X)
```

that solves (1) for $x \in [-2,2]$ by means of finite volumes using the Godunov flux functions and considering non-reflecting Neumann boundary conditions. You may use the simplified formula (4.16) in Exercise 4.1 in the (hyperbolic) lecture notes.

The input arguments are the number of spatial cells to be used N, the final time T, the flux function f ↪ , and the function u0 indicating the initial value. The output arguments are the approximated solution u at time T and the vector of grid points X.

Hint: The (hyperbolic) lecture notes can be found in the lecture's webpage or in the following link: `https://moodle-app2.let.ethz.ch/pluginfile.php/362752/course/section/41409/numcl_notes_HOMEPAGE.pdf`

1d)

(**Core problem**) In the template file scalarconservationlaw.cpp, implement the function

```
void GodunovConvergence( double T, const std::function<double(double)>& f,
                         const std::function<double(double)>& df,
                         const std::function<double(double)>& u0,
                         const std::function<double(double, double)>& uex,
                         const std::string& baseName)
```

that calculates $L^1(D)$ and $L^\infty(D)$ errors for your Godunov scheme with number of cells 100, 200, 400, 800, 1600.

Input arguments T, f and u0 are as in subproblem **1c)**, while the function df returns the derivative of the flux f, the function uex gives the exact solution and baseName is the name to save the computed errors.

1e)

(**Core problem**) In the template file scalarconservationlaw.cpp, implement the function

```
void LaxFriedrichs(int N, double T, const std::function<double(double)>& f,
                   const std::function<double(double)>& u0,
                   Eigen::VectorXd& u,
                   Eigen::VectorXd& X)
```

that solves (1) for $x \in [-2, 2]$ by means of finite volumes with Lax-Friedrichs flux and considering non-reflecting Neumann boundary conditions. The input and output arguments are the same as in subproblem **1c)**.

1f)

(**Core problem**) In the template file scalarconservationlaw.cpp, implement the function

```
void LFConvergence( double T, const std::function<double(double)>& f,
                         const std::function<double(double)>& df,
                         const std::function<double(double)>& u0,
                         const std::function<double(double, double)>& uex,
                         const std::string& baseName)
```

that calculates $L^1(D)$ and $L^\infty(D)$ errors for your Lax-Friedrichs scheme with number of cells 100, 200, 400, 800, 1600. Input arguments are the same as in subproblem **1d)**.

1g)

Test the convergence of the solutions at $t = 0.8$ obtained by the routines `Godunov` and `LaxFriedrichs` for the Burgers' equation (2) with initial conditions $U(x, 0)$ as in subproblem **1a)** (and $D = [-2, 2]$). Which method performs better?

Hint: Use your implementations from the previous subtasks and the exact solutions computed in subproblem **1a)**.

Hint: You can use `plot_convergence.py` to create the plots. This script takes as input the strings baseName and method (so for example, you can run it typing:
`python plot_convergence.py Burgers1 Godunov`).

1h)

Test your routines `Godunov` and `LaxFriedrichs` (implemented in subproblems **1c)** and **1e)**) now with the **Buckley–Leverett** flux function

$$f(U) = \frac{U^2}{U^2 + (1-U)^2} \tag{8}$$

and initial condition

$$U(x,0) = \begin{cases} 0.1 & \text{if } x < 0, \\ 0.9 & \text{if } x > 0. \end{cases} \qquad \text{for } x \in \mathbb{R}, \tag{9}$$

at $t = 0.8$.

Notice that the simplified formula (4.16) proposed in subproblem **1c)** to implement the Godunov flux is a very particular case. Why does it also work for the Buckley–Leverett flux function?

Hint: Check whether the Buckley–Leverett flux verifies the conditions stated in Exercise 4.1 in the (hyperbolic) lecture notes for this case.

Hint: You can use `plot_convergence.py` to create the plots. This script takes as input the string baseName (so for example, you can plot the solution in `uB_G.txt` by typping:
`python plot_sol.py uB_G`).