4 Interpolation [4/35 marks]

For this task, you will perform a cubic spline interpolation on the data provided in the file **in_interp.csv**. First, plot the data (using MATLAB or Excel) in order to get an idea of the behaviour $f(x)$ should have. Next, write C code that uses cubic splines in order to estimate the function $f(x)$ that cuts through the data. You will then use your code to calculate the value(s) of $f(x)$ at $x_o = 3.5$, where x_o is a runtime input parameter. You will implement your method in **interp()**, reading the values of x and $f(x)$ from the file **in_interp.csv** in the following format:

```
x,f(x)
4.440706476030278,4.403874813710879
4.116904962153069,4.403874813710879
.
.
.
```

and outputting the values of the interpolated value to **out_interp.csv** in the following format (up to 6 decimal places).

```
xo,f(xo)
3.500000,0.234655
3.500000,1.107865
```

构成

The output provided above is an example and does not constitute the solution. There may be more than 1 solution possible here and you must be able to identify the correct interval(s) to compute the interpolated value. For the C implementation, you must dynamically allocate space for x, $f(x)$ and write your code such that it can work for a different set of input data and different x_o value. Finally, you must plot the interpolated function using either MATLAB or Excel. How does the interpolated function look compared with the actual datapoints?

5 Differentiation, differential equations [13/35 marks]

The one-dimensional transport of a quantity $\phi(x,t)$ in a fluid moving at a constant velocity u can be described by the following linear advection equation:

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = 0 \,. \tag{10}$$

(a) Write a C program that solves (10) on the interval $0 \le x \le 1$, and for times $0 \le t \le t_{final}$, using the velocity $u = 1$. The spatial interval is to be discretized by $N_x + 1$ points. That is $x_i = i\Delta x$ with $i = 0, 1, 2, ..., N_x$, where the equidistant grid spacing $\Delta x = 1/N_x$.

Equation (10) can also be written as

$$\frac{\partial \phi}{\partial t} = \mathrm{RHS}(\phi) \tag{11}$$

where RHS denotes the 'right-hand-side' of the equation containing the spatial derivative $-u\partial\phi/\partial x$.

Discretize the advection equation (10) by using the so-called "leapfrog" scheme (central difference approximation for both the spatial and temporal derivatives):

时间项

$$\frac{\phi^{n+1} - \phi^{n-1}}{2\Delta t} \approx \mathrm{RHS}(\phi^n) \text{ for } \quad n \ge 2 \tag{12}$$

and

$$\mathrm{RHS}(\phi_i) \approx -u\frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \text{ for } \quad i = 1, ..., N_x - 1 \tag{13}$$

空间项

where n is the time index so that for instance ϕ_i^{n+1} is ϕ value at time level $n+1$ and at point x_i. Δt is the numerical time step for the time integration. Note that in the leapfrog scheme, both ϕ^n

时间循环:

and ϕ^{n-1} are required to update ϕ^{n+1}. Thus, the scheme does not work for the first time iteration ($n = 1$) and instead the forward Euler scheme is to be used:

$$\frac{\phi^{n+1} - \phi^{n}}{\Delta t} \approx \mathrm{RHS}(\phi^{n}) \text{ for } \quad n = 1 \tag{14}$$

The so-called "periodic" boundary condition is to be adopted at the end points. That is, when evaluating the spatial derivatives at $i = 0$ and $i = N_x$, equation (13) becomes:

$$\mathrm{RHS}(\phi_i) = -u\frac{\phi_1 - \phi_{N_x-1}}{2\Delta x} \quad \text{for} \quad i = 0, N_x \tag{15}$$

注:用第1项和最后一项的处理

$\phi_i^{n+1} + CFL(\phi_{i+1}^{n} - \phi_{i-1}^{n}) = \phi_i^{n-1}$

The following initial condition for ϕ shall be used to start your simulation:

$$0 \le x < 0.125 \quad \rightarrow \quad \phi(x, t = 0) = 0$$

$$0.125 \le x \le 0.375 \quad \rightarrow \quad \phi(x, t = 0) = 0.5\Big(1 - \cos\big(8\pi(x - 0.125)\big)\Big)$$

$$0.375 < x \le 1 \quad \rightarrow \quad \phi(x, t = 0) = 0\,.$$

For your code to run stably (i.e. the solution does not "blow up" towards infinity), your time step needs to satisfy the so-called CFL condition:

$$\mathrm{CFL} = u\frac{\Delta t}{\Delta x} \le 1\,. \tag{16}$$

$\Delta t = \frac{CFL \cdot \Delta x}{u}$.

You will implement your routines in the function **advection()**, reading in the values of u, N_x, CFL and t_{final} from **in_advection.csv** in the following format:

```
u,Nx,CFL,t_final
1.0,200,0.50,1.0
```

$N_t = \frac{t_{final} \cdot u \cdot Nx}{CFL}$.

The parameter t_{final} determines the total number of time iterations required (N_t). That is, your program will integrate the advection equation up to time $t_{final} = N_t\Delta t$. When computing N_t for a given t_{final}, round N_t to next higher integer in case $t_{final}/\Delta t$ is not a full integer. Finally, your program shall output the computed solution of ϕ at the end of the temporal iteration (i.e. at time $t = N_t\Delta t$) to the file **out_advection.csv** in the following format (up to 6 decimal places).

```
x,phi
0.000000,0.000000
0.010000,0.000394
0.020000,0.000907
.
.
.
```

(b) Run your code until time $t_{final} = 1.0$ for the following cases:

- Chose two different resolutions Δx, by setting N_x to 100 and 200.
- For each N_x, chose the time step Δt from the CFL condition, using CFL=1, 0.8 and 0.25.

Plot your solutions at $t = 1.0$ (six plots in total) and compare with the exact solution. The exact solution is the same as the initial condition but shifted by $u \cdot t$ in the positive x direction. That is, if the initial condition is given by the function $g(x)$, then the exact solution at time t would be $g(\mathrm{mod}(x - u \cdot t, 1))$ where $\mathrm{mod}(a, b)$ is the modulus function which computes the remainder of a divided by b. How well does the solution agree with the exact solution? Discuss:

- How does the agreement of the numerical prediction with the exact solution depend on the grid resolution Δx?
- How does the agreement of the numerical prediction with the exact solution depend on the CFL number?
- What happens if you chose CFL>1, for example CFL=1.002?

$$\left\{ \frac{\phi_i^{n+1} - \phi_i^{n-1}}{2\Delta t} + u\,\frac{\phi_{i+1}^{n} - \phi_{i-1}^{n}}{2\Delta x} = O(\Delta t^2 + \Delta x^2) \right.$$

天津大学
Tianjin University
300072 TIANJIN, CHINA

忽略高阶小项

$$\phi_i^{n+1} - \phi_i^{n-1} + \underbrace{\frac{u\Delta t}{\Delta x}}_{CFL}\left(\phi_{i+1}^{n} - \phi_{i-1}^{n}\right) = 0$$

第一轮: $\phi_i^1 = \phi_i^0 - \frac{CFL}{2}\left(\phi_{i+1}^0 - \phi_{i-1}^0\right)$

ϕ

在 $i=0$ 和 $i=Nx$ 处

$$\phi_0^1 = \phi_0^0 - \frac{CFL}{2}\left(\phi_1^0 - \phi_{Nx-1}^0\right).$$

$$\phi_{Nx}^1 = \phi_{Nx}^0 - \frac{CFL}{2}\left(\phi_1^0 - \phi_{Nx-1}^0\right)$$

i=0 φ[1][0] = φ[0][0] - CFL/2 (φ[0][1] - φ[0][NX-1]);

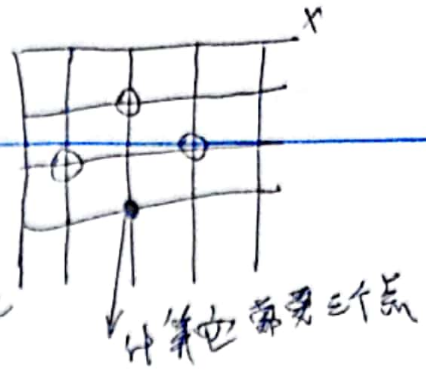
i=其它. for(i=1; i<Nx; i++){
 φ[1][i] = φ[0][i] - CFL/2 (φ[0][i+1] - φ[0][i-1])
}

i=Nx. φ[1][NX] = φ[0][NX] - CFL/2 (φ[0][1] - φ[0][NX-1]);

Scanned with CamScanner

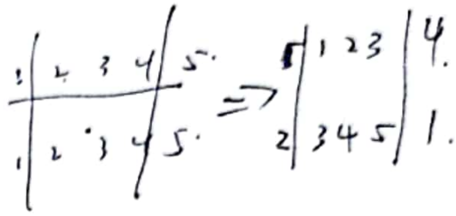
第二轮开始.

在 $i=0$ 和 $i=Nx$ 处.



$$\phi_0^{n+1} = \phi_0^{n-1} - CFL(\phi_1^n - \phi_{Nx-1}^n)$$

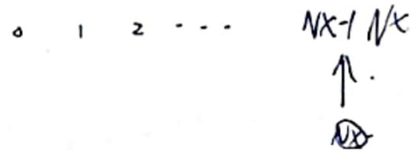
$$\phi_{Nx}^{n+1} = \phi_{Nx}^{n-1} - CFL(\phi_1^n - \phi_{Nx-1}^n)$$



在 $i=$ 其它.

$$\phi_i^{n+1} = \phi_i^{n-1} - CFL(\phi_{i+1}^n - \phi_{i-1}^n)$$

$n=0$ 即初始化、$n=1$ 才是一步



第二轮迭代开始:

```
for(n=1; n<Nt-1; n++){
i=0处: φ[n+1][0] = φ[n-1][0] - CFL(φ[n][1] - φ[n][Nx-1]);
i=其它: for(i=1; i<Nx; i++){
        φ[n+1][i] = φ[n-1][i] - CFL(φ[n][i+1] - φ[n][i-1]);
        }
i=Nx处: φ[n+1][Nx] = φ[n-1][Nx] - CFL(φ[n][1] - φ[n][Nx-1]);
```