MAE 4990: CFD Training - Module 0

In this course, you will learn the Finite Volume method as your first step into CFD. The course is organized into a sequence of modules spanning topics from the 1D linear wave equation to the 2D compressible Navier–Stokes equations.

Module Goal

The purpose of this module is to familiarize you with the time integrators used throughout the course, basic grid generation and file I/O (read/write), and the computation of grid geometry. You will Implement the reusable infrastructure for all later CFD modules:

1. SSPRK2 and SSPRK3 time stepping for the semi-discrete system

$$\frac{d\bar{U}}{dt} = \mathcal{L}(\bar{U}, t), \tag{1}$$

   where $\bar{U}$ denotes cell-averaged unknowns and $\mathcal{L}(\cdot, t)$ is the residual operator.

2. Reading structured grids (1D/2D) from file and computing geometric quantities needed by FV residual assembly: cell centers, face vectors, and cell sizes/areas.

1 Strong Stability-Preserving Runge–Kutta (SSPRK) Time Integrators

Throughout the course, you will discretize the spatial terms of the governing equations to obtain the semi-discrete system

$$\frac{d\bar{U}}{dt} = \mathcal{L}(\bar{U}, t). \tag{2}$$

Whether you are solving linear advection or the full compressible Navier–Stokes equations, you will always arrive at this same ODE form. For that reason, you will first implement a *generic* time integrator that can be reused across different governing equations.

Here, $\mathcal{L}(\cdot, t)$ represents the spatially discretized operator, including convective, viscous, and source terms (when applicable). We refer to $\mathcal{L}$ as the *residual* or the *right-hand side (RHS)*. In code, this is typically implemented as

$$R = \mathcal{L}(\bar{U}, t) \qquad \Longrightarrow \qquad \texttt{R = rhs(U, t, geom, params)}. \tag{3}$$

Here, `U` is the current solution, `t` is the current time, `geom` stores the grid/geometry information, and `params` stores implementation-specific parameters. Note that, depending on the application, `rhs` may require additional input arguments (e.g., boundary-condition data, exact-solution functions for manufactured solutions, or model parameters). You should therefore treat the notation above as representative of a generic implementation rather than a fixed function signature.

1.1 SSPRK as Forward Euler Steps

The SSPRK schemes used in this course are conveniently expressed in terms of a single Forward Euler step operator,

$$\mathcal{F}\big(\bar{U}, t, \Delta t\big) \;=\; \bar{U} + \Delta t\, \mathcal{L}(\bar{U}, t). \tag{4}$$

SSPRK methods can be written as convex combinations of such Forward Euler steps applied to intermediate stage solutions. In practice, this means that each stage consists of (i) a single evaluation of the RHS routine and (ii) an update of the intermediate solution using the SSPRK coefficients.

1.2 SSPRK2 (second order)

Given $(\bar{U}^n, t^n)$ and timestep Δt:

$$\bar{U}^{(1)} = \mathcal{F}\big(\bar{U}^n,\, t^n,\, \Delta t\big)\,, \tag{5}$$

$$\bar{U}^{n+1} = \tfrac{1}{2}\bar{U}^n + \tfrac{1}{2}\,\mathcal{F}\Big(\bar{U}^{(1)},\, t^n + \Delta t,\, \Delta t\Big)\,, \tag{6}$$

then set $t^{n+1} = t^n + \Delta t$. This method calls the RHS routine twice per timestep (once at each stage) and combines the stage updates using the SSPRK2 coefficients.

1.3 SSPRK3 (third order)

Given $(\bar{U}^n, t^n)$ and timestep Δt:

$$\bar{U}^{(1)} = \mathcal{F}\big(\bar{U}^n,\, t^n,\, \Delta t\big)\,, \tag{7}$$

$$\bar{U}^{(2)} = \tfrac{3}{4}\bar{U}^n + \tfrac{1}{4}\,\mathcal{F}\Big(\bar{U}^{(1)},\, t^n + \Delta t,\, \Delta t\Big)\,, \tag{8}$$

$$\bar{U}^{n+1} = \tfrac{1}{3}\bar{U}^n + \tfrac{2}{3}\,\mathcal{F}\Big(\bar{U}^{(2)},\, t^n + \tfrac{1}{2}\Delta t,\, \Delta t\Big)\,, \tag{9}$$

then set $t^{n+1} = t^n + \Delta t$. This method calls the RHS routine three times per timestep (once per stage) and combines the stage updates using the SSPRK3 coefficients.

2 Grid Files (Structured Nodes)

In this course, you will implement the Finite Volume method on structured 1D/2D grids. (The same ideas extend to unstructured grids, but we will not cover them here.)

A finite-volume solver starts from *node coordinates* and builds the geometric information it needs for flux balances. In practice, the grid is typically generated *outside* the solver using a separate tool or script, written to a file (text or binary), and then read by the solver at runtime. During a preprocessing stage, the solver computes the required geometric data (e.g., cell centers, face vectors, cell lengths/areas/volumes). Only after this geometry is available does the solver proceed to apply the numerical method and advance the solution in time.

Here, we will first examine what a grid file looks like in one dimension and how to write and read it. We will then move on to a two-dimensional example.

2.1 1D Grid File

Format

- Line 1: `Nx` (number of nodes)

- Lines 2..Nx+1: `x(i)` for $i = 1, \ldots, N_x$

Cell i spans $[x_i, x_{i+1}]$ for $i = 1, \ldots, N_x - 1$.

Example: 10 nodes on $[0, 1]$

File `grid1d_10nodes.txt`:

```
10
0.0
0.1111111111
0.2222222222
0.3333333333
0.4444444444
0.5555555556
0.6666666667
0.7777777778
0.8888888889
1.0
```

MATLAB read/write example

```
% --- write example 1D grid file (10 nodes on [0,1]) ---
Nx = 10;
x = linspace(0,1,Nx).';
fid = fopen("grid1d_10nodes.txt","w");
fprintf(fid,"%d\n",Nx);
fprintf(fid,"%.10f\n",x);
fclose(fid);

% --- read 1D grid file ---
fid = fopen("grid1d_10nodes.txt","r");
Nx = fscanf(fid,"%d",1);
x = fscanf(fid,"%f",Nx);
fclose(fid);
```

2.2 2D Grid File (Structured Nodes)

Format

- Line 1: `Nx Ny` (number of nodes in x and y directions)
- Next `Nx*Ny` lines: `x(i,j) y(i,j)` in ordering: j outer loop, i inner loop.

Example: $N_x = 4$, $N_y = 3$ nodes on a rectangle

File `grid2d_4x3nodes.txt` (here $x = i - 1$, $y = j - 1$):

```
4 3
0.0 0.0
1.0 0.0
2.0 0.0
3.0 0.0
```

```
0.0 1.0
1.0 1.0
2.0 1.0
3.0 1.0
0.0 2.0
1.0 2.0
2.0 2.0
3.0 2.0
```

MATLAB read example (matching the ordering above)

```
fid = fopen("grid2d_4x3nodes.txt","r");
Nx = fscanf(fid,"%d",1);
Ny = fscanf(fid,"%d",1);

x = zeros(Nx,Ny); y = zeros(Nx,Ny);
for j = 1:Ny
  for i = 1:Nx
    x(i,j) = fscanf(fid,"%f",1);
    y(i,j) = fscanf(fid,"%f",1);
  end
end
fclose(fid);
```

3 Geometry Preprocessing

Once a grid file is available and has been read by the solver, the next step is to compute the required geometric properties. Because the grid file contains only node coordinates, the preprocessing stage must derive quantities such as cell volumes/areas, face measures, and cell/face distances from the node data.

3.1 1D Geometry from Nodes

The grid file provides *node* coordinates, but in a finite-volume solver we store the *cell-averaged* solution at *cell centers*. For this reason, we use two related index sets:

- **Cell interfaces (faces):** $x_{i\pm1/2}$,
- **Cell centers:** x_i.

Do not confuse this solver notation with the indexing used by a grid writer/reader. In this course, within the solver, cell interfaces are always labeled with half-indices ($i \pm 1/2$) and cell centers are labeled with integer indices (i). With this convention, cell i spans the interval $\left[x_{i-1/2},\, x_{i+1/2}\right]$.

Given the cell interface locations $x_{i\pm1/2}$, the cell center and cell width are

$$x_i = \tfrac{1}{2}\left(x_{i+1/2} + x_{i-1/2}\right), \qquad \Delta x_i = x_{i+1/2} - x_{i-1/2}, \qquad i = 1, \ldots, N_x, \tag{10}$$

where N_x is the number of cells in the x-direction. Note that the number of cells in a given direction is one less than the number of grid nodes in that direction:

$$N_x = (\#\text{ of nodes}) - 1.$$

This also means that the domain boundaries coincide with cell interfaces adjacent to the boundary, which are the locations where boundary conditions are imposed.

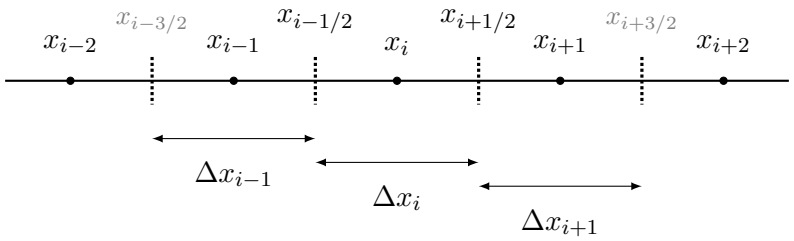

Figure 1: 1D finite-volume indexing: cell centers x_i and cell faces $x_{i\pm1/2}$.

3.2 2D Geometry from Structured Quad Nodes

In the solver, we index *cells* by (i, j):

$$i = 1, \ldots, N_x, \qquad j = 1, \ldots, N_y,$$

so the domain contains $N_x \times N_y$ cells. The grid file, however, stores *node* coordinates on a different index set. We use capital indices (I, J) for nodes:

$$I = 1, \ldots, N_x + 1, \qquad J = 1, \ldots, N_y + 1,$$

so there are $(N_x + 1) \times (N_y + 1)$ nodes. The node coordinates read from the file are

$$\mathbf{x}_{I,J} = (x_{I,J}, y_{I,J}).$$

The use of capital letters (I, J) is intentional: node indices are not the same objects as cell indices (i, j).

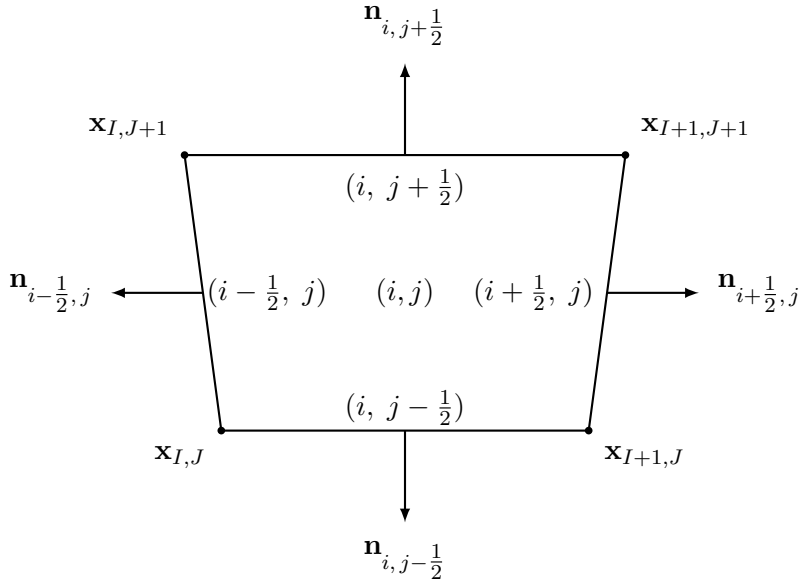

Figure 2: Cell (i, j): corner nodes $x_{i,j}$, face indices at half-steps, and outward unit normals at faces.

Cell (i, j) is bounded by four corner nodes:

$$\mathbf{x}_1 = \mathbf{x}_{I,J}, \quad \mathbf{x}_2 = \mathbf{x}_{I+1,J}, \quad \mathbf{x}_3 = \mathbf{x}_{I+1,J+1}, \quad \mathbf{x}_4 = \mathbf{x}_{I,J+1},$$

where (I, J) denotes the index of the lower-left corner node of the cell, and $\mathbf{x}_m = (x_m, y_m)$. As in 1D, we label faces using half-indices relative to the cell:

$$(i - \tfrac{1}{2},\, j) \text{ (left)}, \qquad (i + \tfrac{1}{2},\, j) \text{ (right)}, \qquad (i,\, j - \tfrac{1}{2}) \text{ (bottom)}, \qquad (i,\, j + \tfrac{1}{2}) \text{ (top)}.$$

Cell area. Using the corner coordinates $\mathbf{x}_m = (x_m, y_m)$ for $m = 1, 2, 3, 4$,

$$|K_{i,j}| = \frac{1}{2} \left| \sum_{m=1}^{4} (x_m y_{m+1} - y_m x_{m+1}) \right|, \tag{11}$$

where $\mathbf{x}_5 = \mathbf{x}_1$ (equivalently, $(x_5, y_5) = (x_1, y_1)$) closes the loop.

Outward unit normal vectors at faces Let the cell corners be ordered consecutively around the cell boundary, and let face m connect $\mathbf{x}_m \to \mathbf{x}_{m+1}$. Define the face tangent

$$\mathbf{t}_m = \mathbf{x}_{m+1} - \mathbf{x}_m = (t_x, t_y),$$

and let f_m denote face m. Its length is

$$|f_m| = \|\mathbf{t}_m\| = \sqrt{t_x^2 + t_y^2}.$$

The outward unit normal vector is defined by

$$\mathbf{n}_m = \frac{1}{|f_m|} \begin{bmatrix} t_y \\ -t_x \end{bmatrix}.$$

Store the four face vectors $\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3, \mathbf{n}_4$ for each cell.

4 Required MATLAB Files and Interfaces

Time Steppers

- `U = ssprk2(U, t, dt, rhs, geom, params)`
- `U = ssprk3(U, t, dt, rhs, geom, params)`

Grid and Geometry

- `grid = read_grid_1d(filename)` (returns node array `x`)
- `grid = read_grid_2d(filename)` (returns node arrays `x(i,j)`, `y(i,j)`)
- `geom = build_geom_1d(grid)` (returns `xc`, `dx`)
- `geom = build_geom_2d(grid)` (returns `xc`, `yc`, `A`, face vectors per cell)

Residual Handle

Your steppers must call a residual of the form:

$$\texttt{R = rhs(U, t, geom, params).} \tag{12}$$

5 MATLAB Examples to Complete

Example A: Nonautonomous ODE (stage-time correctness)

Solve

$$u'(t) = -u(t) + \sin(t), \qquad u(0) = 0, \qquad t \in [0, T], \tag{13}$$

with $T = 10$. The exact solution is

$$u(t) = \tfrac{1}{2}\left(\sin t - \cos t\right) + \tfrac{1}{2}e^{-t}. \tag{14}$$

Implement `rhs_ode(u,t)` and use your SSPRK2 and SSPRK3 implementations with multiple timestep sizes Δt. For each run, (i) plot $u(t)$ over $t \in [0, T]$ and overlay the exact solution, and (ii) report the final-time error $|u(T) - u_{\text{exact}}(T)|$.

In addition, compare your results to MATLAB's built-in ODE solvers (e.g., `ode45`) on the same time interval.

MATLAB example (reference).

```
T = 10;
rhs_ode = @(t,u) -u + sin(t);

% Reference solution with ode45
[t_ref,u_ref] = ode45(rhs_ode,[0 T],0);

% Exact solution for plotting
u_exact = 0.5*(sin(t_ref) - cos(t_ref)) + 0.5*exp(-t_ref);

figure; hold on; box on;
plot(t_ref,u_ref,"k-","DisplayName","ode45");
plot(t_ref,u_exact,"r--","DisplayName","Exact");
xlabel("t"); ylabel("u(t)"); legend("Location","best");
```

Example B: 2D Grid Preprocessing and Visualization

Read a small 2D structured grid file (node coordinates), then compute and store the 2D geometric quantities required by later modules (cell centers, cell areas, and face normals/face measures or face vectors).

To visualize the grid *directly from node coordinates* (X_g, Y_g), you may plot grid lines along the two index directions, for example:

```
figure; hold on; axis equal; box on;

% Plot grid lines along rows (constant i)
for i = 1:size(Xg, 1)
    plot(Xg(i, :), Yg(i, :), 'k-');
end
```

```
% Plot grid lines along columns (constant j)
for j = 1:size(Xg, 2)
    plot(Xg(:, j), Yg(:, j), 'k-');
end

xlabel('x'); ylabel('y');
title('Structured grid (nodes)');
```

On the same figure, overlay cell centers (from your preprocessor) and, for a small subset of cells, visualize the outward face normals (or face vectors) to confirm consistency.

6 Submission

Submit a short, well-formatted **PDF report** that includes your results for Examples A and B (figures, brief captions, and any requested error values). You may choose your own report format and styling. This report will become part of your end-of-semester compilation.