Prof. Fabien Evrard & Fabian Dettenrieder
AE 410/CSE 461 – Computational Aerodynamics
Spring 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Take-home Midterm

Due date: March 22, 2024

The objective of this take-home midterm is to write a finite-difference solver for the inviscid flow in a 1D duct. This flow is governed by the 1D Euler equations, which are given in their conservative form as

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0 \tag{1}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial}{\partial x}(\rho u^2 + p) = 0 \tag{2}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial}{\partial x}[(\rho E + p)u] = 0 \tag{3}$$

where

- ρ is the fluid density,
- u is the fluid velocity,
- p is the thermodynamic pressure,
- E is the total specific energy.

The pressure is related to the other quantities by the equation of state

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho u^2\right) \tag{4}$$

for a perfect gas with ratio of specific heats γ.

The Euler equations (1)–(3) constitute a system of coupled nonlinear partial differential equations, which is typically very difficult to solve analytically. Instead, in this midterm, we will solve this system of equations numerically using finite differences and the method of lines on a periodic domain.

Problem 1 · The Euler equations (1)–(3) are written in terms of the conserved variables that are: the mass density ρ (mass per unit volume), the momentum density ρu (momentum per unit volume), and the total energy density ρE (total energy per unit volume). Let us assign these conserved variables to a single vector

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix} \tag{5}$$

Q1 → Using the equation of state (4), express the pressure as a function of the components of $\mathbf{q}$, i.e., as a function of q_0, q_1, and q_2.

Hint: For instance, the fluid velocity u can be expressed as

$$u = \frac{q_1}{q_0} \tag{6}$$

Solution: Using the EOS, we can write pressure as a function of the conservative variables as

$$\begin{aligned} p &= (\gamma - 1)\left[\rho E - \frac{1}{2}\rho u^2\right] \\ &= (\gamma - 1)\left[q_2 - \frac{1}{2}\frac{q_1^2}{q_0}\right] \end{aligned}$$

Problem 2 · The Euler equations (1)–(3) may be written in vector form as

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial x} = \mathbf{0} \tag{7}$$

where $\mathbf{f}(\mathbf{q})$ is the flux vector given as

$$\mathbf{f}(\mathbf{q}) = \begin{bmatrix} f_0(\mathbf{q}) \\ f_1(\mathbf{q}) \\ f_2(\mathbf{q}) \end{bmatrix} \tag{8}$$

Q2 → Write $\mathbf{f}(\mathbf{q})$ in terms of the components of $\mathbf{q}$.

Hint: For instance, it is pretty clear from Eq. (1) that

$$f_0(\mathbf{q}) = q_1 \tag{9}$$

Solution: For the flux vector, we know that

$$\begin{aligned} f_0 &= \rho u \\ f_1 &= \rho u^2 + p \\ f_2 &= (\rho E + p)\, u \end{aligned}$$

which we can rewrite using conservative variables as

$$\begin{aligned} f_0 &= q_1 \\ f_1 &= (\gamma - 1)\, q_2 + \frac{1}{2}(3 - \gamma)\frac{q_1^2}{q_0} \\ f_2 &= \gamma \frac{q_1 q_2}{q_0} - \frac{1}{2}(\gamma - 1)\frac{q_1^3}{q_0^2} \end{aligned}$$

Problem 3 · Now that we have expressed the Euler equations (1)–(3) in vector form, it is useful to express them in the quasi-linear form

$$\frac{\partial \mathbf{q}}{\partial t} + \mathbf{M}\frac{\partial \mathbf{q}}{\partial x} = \mathbf{0} \tag{10}$$

where $\mathbf{M}$ is a 3×3 flux-Jacobian matrix defined as

$$\mathbf{M}(\mathbf{q}) = \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} = \begin{bmatrix} \dfrac{\partial f_0(\mathbf{q})}{\partial q_0} & \dfrac{\partial f_0(\mathbf{q})}{\partial q_1} & \dfrac{\partial f_0(\mathbf{q})}{\partial q_2} \\ \dfrac{\partial f_1(\mathbf{q})}{\partial q_0} & \dfrac{\partial f_1(\mathbf{q})}{\partial q_1} & \dfrac{\partial f_1(\mathbf{q})}{\partial q_2} \\ \dfrac{\partial f_2(\mathbf{q})}{\partial q_0} & \dfrac{\partial f_2(\mathbf{q})}{\partial q_1} & \dfrac{\partial f_2(\mathbf{q})}{\partial q_2} \end{bmatrix} \tag{11}$$

Q3 → Find the expression of the matrix $\mathbf{M}$.

Remark: This matrix is **not** the same as the flux-Jacobian we have derived in class for the 1D Euler equations in non-conservative form. In class, we have considered the variables $(q_0, q_1, q_2) = (\rho, u, p)$; in this midterm, we consider the conserved variables $(q_0, q_1, q_2) = (\rho, \rho u, \rho E)$.

Solution: Rewriting the Euler equations in the quasi-linear form

$$\frac{\partial \mathbf{q}}{\partial t} + \mathbf{M}(\mathbf{q}) \frac{\partial \mathbf{q}}{\partial x} = 0, \tag{12}$$

we get the sought-after flux Jacobian as a function of conservative variables as

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma - 3)\frac{q_1^2}{q_0^2} & (3-\gamma)\frac{q_1}{q_0} & (\gamma - 1) \\ -\gamma \frac{q_1 q_2}{q_0^2} + (\gamma - 1)\frac{q_1^3}{q_0^3} & \gamma \frac{q_1}{q_0} - \frac{3}{2}(\gamma - 1)\frac{q_1^2}{q_0^2} & \gamma \frac{q_1}{q_0} \end{bmatrix} \tag{13}$$

Problem 4 · The Jacobian matrix $\mathbf{M}$ admits a complete set of eigenvalues and eigenvectors $(\Omega_n, \mathbf{X}_n), n \in \{0, 1, 2\}$, such that

$$\mathbf{M}\mathbf{X}_n = \Omega_n \mathbf{X}_n \ , \quad \forall n \in \{0, 1, 2\} \tag{14}$$

which leads to the eigendecomposition

$$\mathbf{M} = \mathbf{X}\mathbf{\Omega}\mathbf{X}^{-1} \tag{15}$$

with

$$\mathbf{\Omega} = \mathrm{diag}\,(\Omega_0, \Omega_1, \Omega_2) \tag{16}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_0 & \mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix} \tag{17}$$

Q4 → Find the eigenvalues and eigenvectors of $\mathbf{M}$ and order them such that $\Omega_0 \leq \Omega_1 \leq \Omega_2$.

Hint: To keep the expressions relatively compact, it is useful to introduce the speed of sound c given by

$$c = \sqrt{\frac{\gamma p}{\rho}} \tag{18}$$

Solution: Given the flux Jacobian from Problem 3, we can rewrite it in terms of primitive variables as

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^2 & (3-\gamma)u & (\gamma-1) \\ -\gamma u E + (\gamma-1)u^3 & \gamma E - \frac{3}{2}(\gamma-1)u^2 & \gamma u \end{bmatrix} \tag{19}$$

where we can make use of the total specific enthalpy

$$H = E + \frac{p}{\rho} = \frac{c^2}{\gamma-1} + \frac{1}{2}u^2 \tag{20}$$

to simplify it to

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^2 & (3-\gamma)u & (\gamma-1) \\ \frac{1}{2}(\gamma-2)u^3 - \frac{c^2 u}{\gamma-1} & \frac{c^2}{\gamma-1} - \left(\gamma - \frac{3}{2}\right)u^2 & \gamma u \end{bmatrix}. \tag{21}$$

Using a symbolic toolbox, we can solve for the eigenvalues and eigenvectors to get

$$\Omega_0 = u - c \qquad \Omega_1 = u \qquad \Omega_2 = u + c$$

$$\mathbf{X}_0 = \begin{bmatrix} 1 \\ u-c \\ H - uc \end{bmatrix} \qquad \mathbf{X}_1 = \begin{bmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{bmatrix} \qquad \mathbf{X}_3 = \begin{bmatrix} 1 \\ u+c \\ H + uc \end{bmatrix}$$

Problem 5 · We will use a flux-splitting strategy to solve the system of equations (10). As a result, we need to construct finite-difference operators that are biased in the positive and negative directions. We choose to use the finite-difference operators that approximate a given function $f(x)$ as

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_n^+ = \sum_{j=-3}^{1} a_j^+ f_{n+j} \tag{22}$$

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_n^- = \sum_{j=-1}^{3} a_j^- f_{n+j} \tag{23}$$

In other terms, $\mathrm{d}\cdot/\mathrm{d}x|^+$ will be upwind if information propagates from left to right, and $\mathrm{d}\cdot/\mathrm{d}x|^-$ will be upwind is information propagates from right to left.

Q5.1 → Find the coefficients $\{a_j^+\}_{j=-3}^1$ and $\{a_j^-\}_{j=-1}^3$ that lead to the finite-difference approximations (22) and (23).

Q5.2 → Derive the leading truncation error term for each scheme. What is the order of accuracy of these two finite-difference schemes?

Solution: To find the coefficients a_j^-, we use Taylor series approximations about f_i as follows:

$$
\begin{aligned}
f_{i-1} &= f_i - \Delta x \frac{df}{dx}\bigg|_{x_i} + \frac{\Delta x^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_i} - \frac{\Delta x^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_i} + \frac{\Delta x^4}{4!}\frac{d^4f}{dx^4}\bigg|_{x_i} + \text{h.o.t.} \\
f_i &= f_i \\
f_{i+1} &= f_i + \Delta x \frac{df}{dx}\bigg|_{x_i} + \frac{\Delta x^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_i} + \frac{\Delta x^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_i} + \frac{\Delta x^4}{4!}\frac{d^4f}{dx^4}\bigg|_{x_i} + \text{h.o.t.} \\
f_{i+2} &= f_i + (2\Delta x) \frac{df}{dx}\bigg|_{x_i} + \frac{(2\Delta x)^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_i} + \frac{(2\Delta x)^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_i} + \frac{(2\Delta x)^4}{4!}\frac{d^4f}{dx^4}\bigg|_{x_i} + \text{h.o.t.} \\
f_{i+3} &= f_i + (3\Delta x) \frac{df}{dx}\bigg|_{x_i} + \frac{(3\Delta x)^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_i} + \frac{(3\Delta x)^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_i} + \frac{(3\Delta x)^4}{4!}\frac{d^4f}{dx^4}\bigg|_{x_i} + \text{h.o.t.}
\end{aligned}
$$

Using a Taylor Table approach to solve for the stencils, we get the linear system of equations

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
-\Delta x & 0 & \Delta x & 2\Delta x & 3\Delta x \\
\Delta x^2 & 0 & \Delta x^2 & \left(\Delta x^2\right)^2 & \left(2\Delta x^2\right)^2 \\
-\Delta x^3 & 0 & \Delta x^3 & \left(\Delta x^2\right)^3 & \left(2\Delta x^2\right)^3 \\
\Delta x^4 & 0 & \Delta x^4 & \left(\Delta x^2\right)^4 & \left(2\Delta x^2\right)^4
\end{bmatrix}
\begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{24}
$$

which we solve using the symbolic toolbox within Matlab and get the stencils

$$
a^- = \frac{1}{12\Delta x}\begin{bmatrix} -3 \\ -10 \\ 18 \\ -6 \\ 1 \end{bmatrix}. \tag{25}
$$

Similarly, we can solve for the other scheme, which uses the same number of points but is biased in the opposite direction and leads to

$$
a^+ = \frac{1}{12\Delta x}\begin{bmatrix} -1 \\ 6 \\ -18 \\ 10 \\ 3 \end{bmatrix} \tag{26}
$$

The leading truncation error for the stencils follow

$$
\begin{aligned}
\epsilon^- &= \frac{\Delta x^4}{20}\frac{\partial^5 f}{\partial x^5}\bigg|_i \\
\epsilon^+ &= \frac{\Delta x^4}{20}\frac{\partial^5 f}{\partial x^5}\bigg|_i
\end{aligned}
$$

and the schemes are thus 4^{th} order accurate.

Problem 6 · We want to know whether the schemes constructed in Problem 5 are dispersive and/or dissipative.

Q6.1 → Using Fourier error analysis, derive the expression of the modified wavenumber $\kappa^\star$ associated with each scheme.

Q6.2 → Plot the real and imaginary parts of $\kappa^\star \Delta x$ as a function of $\kappa \Delta x$ between 0 and π, and explain if one should expect the schemes to be dispersive and/or dissipative.

Solution: In order to quantify the dispersive and dissipative behavior of the two schemes at hand, we utilize the Fourier error analysis, as follows.

$$\begin{aligned}
\left.\frac{\partial f}{\partial x}\right|_n^- &= \frac{1}{12\Delta x}\left[-3f_{n-1} - 10f_n + 18f_{n+1} - 6f_{n+2} + f_{n+3}\right] \\
&= \frac{e^{i\kappa x}}{12\Delta x}\left[-3e^{-i\kappa\Delta x} - 10 + 18e^{i\kappa\Delta x} - 6e^{2i\kappa\Delta x} + e^{3i\kappa\Delta x}\right] \\
&= \frac{e^{i\kappa x}}{12\Delta x}\Big[\Big[15\cos(\kappa\Delta x) - 6\cos(2\kappa\Delta x) + \cos(3\kappa\Delta x) - 10\Big] \\
&\qquad + i\Big[21\sin(\kappa\Delta x) - 6\sin(2\kappa\Delta x) + \sin(3\kappa\Delta x)\Big]\Big]
\end{aligned}$$

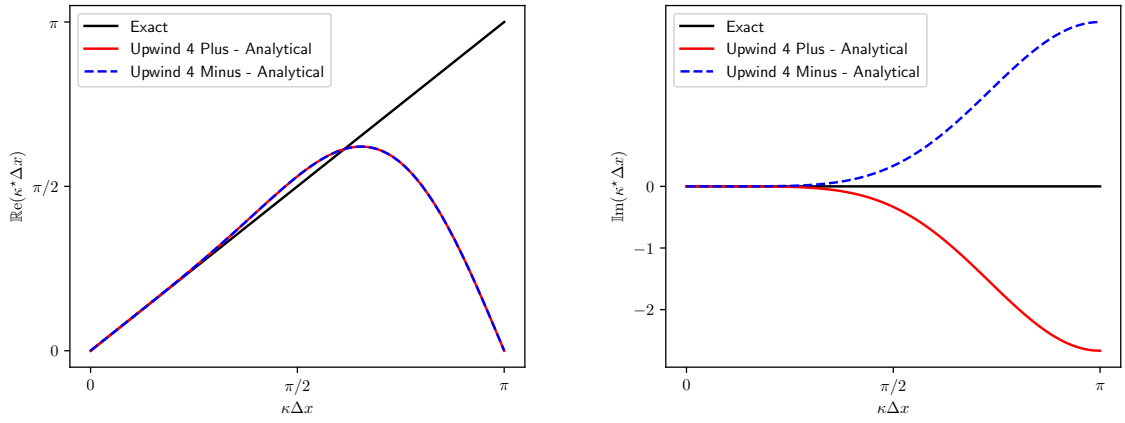
Consequently, the modified wave number follows the expression

$$\kappa^\star \Delta x\big|^- = \frac{1}{12}\Big[\Big[21\sin(\kappa\Delta x) - 6\sin(2\kappa\Delta x) + \sin(3\kappa\Delta x)\Big] - i\Big[15\cos(\kappa\Delta x) - 6\cos(2\kappa\Delta x) + \cos(3\kappa\Delta x) - 10\Big]\Big]$$

Equally, we can derive the modified wave number for the second scheme as

$$\kappa^\star \Delta x\big|^+ = \frac{1}{12}\Big[\Big[21\sin(\kappa\Delta x) - 6\sin(2\kappa\Delta x) + \sin(3\kappa\Delta x)\Big] - i\Big[-15\cos(\kappa\Delta x) + 6\cos(2\kappa\Delta x) - \cos(3\kappa\Delta x) + 10\Big]\Big]$$

As the plot below shows, the two schemes have little dispersion for a wide range of frequencies. The imaginary part of the the modified wavenumber shows dispersive behavior, where the D^+ is stable but dissipative and D^- is unstable for the linear advection equation (with for $a > 0$). For the case of $a < 0$, the behavior is switch and D^- is stable but dissipative and D^+ is unstable.



Problem 7 · We want to solve the system of equations (10) in a stable manner by splitting the flux-Jacobian matrix $\mathbf{M}$ as

$$\mathbf{M} = \underbrace{\mathbf{X}\boldsymbol{\Omega}^+\mathbf{X}^{-1}}_{\mathbf{M}^+} + \underbrace{\mathbf{X}\boldsymbol{\Omega}^-\mathbf{X}^{-1}}_{\mathbf{M}^-} \tag{27}$$

with

$$\mathbf{\Omega}^+ = \frac{1}{2}\left(\mathbf{\Omega} + |\mathbf{\Omega}|\right) \tag{28}$$

$$\mathbf{\Omega}^- = \mathbf{\Omega} - \mathbf{\Omega}^+ \tag{29}$$

This leads to the system of equations

$$\frac{\partial \mathbf{q}}{\partial t} + \mathbf{M}^+ \frac{\partial \mathbf{q}}{\partial x} + \mathbf{M}^- \frac{\partial \mathbf{q}}{\partial x} = \mathbf{0} \tag{30}$$

Following the method of lines and making use of our knowledge of the sign of $\mathbf{\Omega}^+$ and of $\mathbf{\Omega}^-$, we approximate the spatial derivative in the term $\mathbf{M}^+ \partial \mathbf{q}/\partial x$ with the biased scheme given in Eq. (22), and the spatial derivative in the term $\mathbf{M}^- \partial \mathbf{q}/\partial x$ with the biased scheme given in Eq. (23), so as to transform Eq. (30) into a system of ordinary differential equations (i.e., the semi-discrete 1D Euler problem).

Q7 → Develop a code for solving the semi-discrete 1D Euler problem on the periodic domain $[0, L]$ discretized into N_x grid points.

Hints: You may reuse parts of the code that was developed in HW3 for the semi-discrete 1D linear advection problem. Remember that in this midterm, we consider the three conserved variables $(\rho, \rho u, \rho E)$, therefore there will be three discrete variables at each grid point (i.e., $3N_x$ discrete variables in total). You may order the vector of discrete variables in whichever way you see fit; this should not affect your end results.

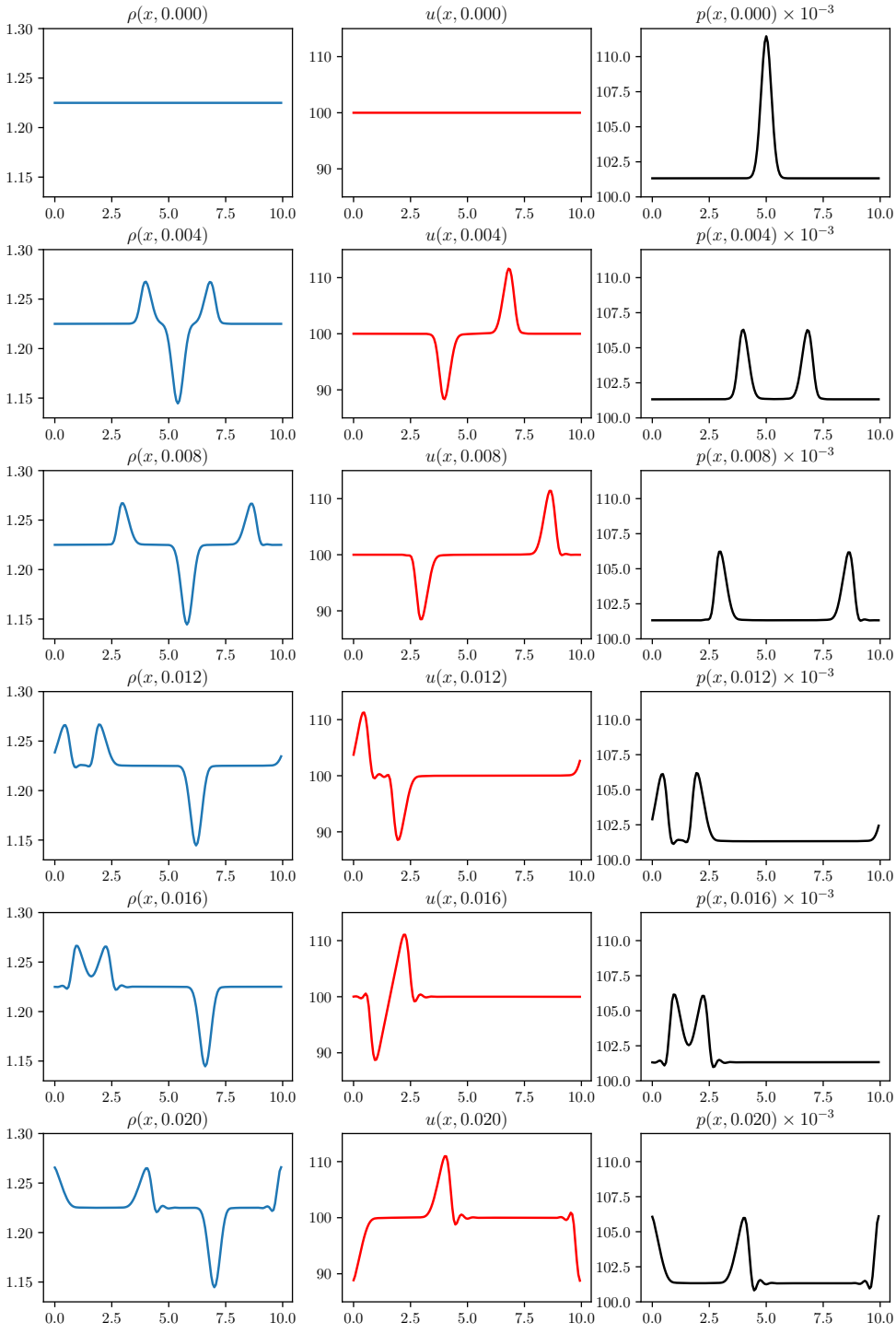
Solution: The code can be found in the Appendix.

Problem 8 · Solve the semi-discrete 1D Euler periodic problem with:

- $L = 10$ m, $N_x = 150$
- $\gamma = 1.4$
- $\rho(x, t = 0) = 1.225$ kg/m^3
- $u(x, t = 0) = 100$ m/s
- $p(x, t = 0) = p_\infty \left(1 + \frac{1}{10} \exp\left(-10(x - L/2)^2\right)\right)$, with $p_\infty = 101325$ Pa.

Q8 → Plot the spatial variations of (ρ, u, p) at 4 millisecond intervals between $t = 0$ and $t = 0.02$ s, and explain what you observe.

Solution: Lastly, we solve the Euler equations with the given initial condition of a Gaussian pressure pulse which triggers all three characteristics. The unsteady solution for all three primitive variables ρ, u, and p is shown in the figures below. We can clearly see three flow phenomena that consist of two downstream propagating and one upstream propagating characteristic, each with its individual advection speed corresponding to its eigenvalue. As expected, treating each characteristic individually with an upwind scheme results in a stable solution for the entire system and the incorporated numerical dissipation helps reduce any non-physical, numerical oscillations around sharp gradients. At later time steps, smaller such oscillations can be observed which arise from the dispersive properties of the numerical scheme.



Problem 9 · **This problem is required for students taking AE 410/CSE 461 for four credit hours. It is not required for students taking AE 410/CSE 461 for three credit hours.**

It is customary in CFD development to verify ones code before using it, so as to guarantee the correctness of the simulation results. There exist several ways to do so, including:

Option 1: Develop an exact solution to the governing equations using the method of characteristics, and compare your numerical solution against this analytical solution.

Option 2: Research and implement the method of manufactured solutions.

Option 3: Compare your numerical solution against the numerical solution of another, already established code (this is called cross-code verification).

Q9 → Verify the code you have developed for Problems 7/8 using one of the listed approaches.

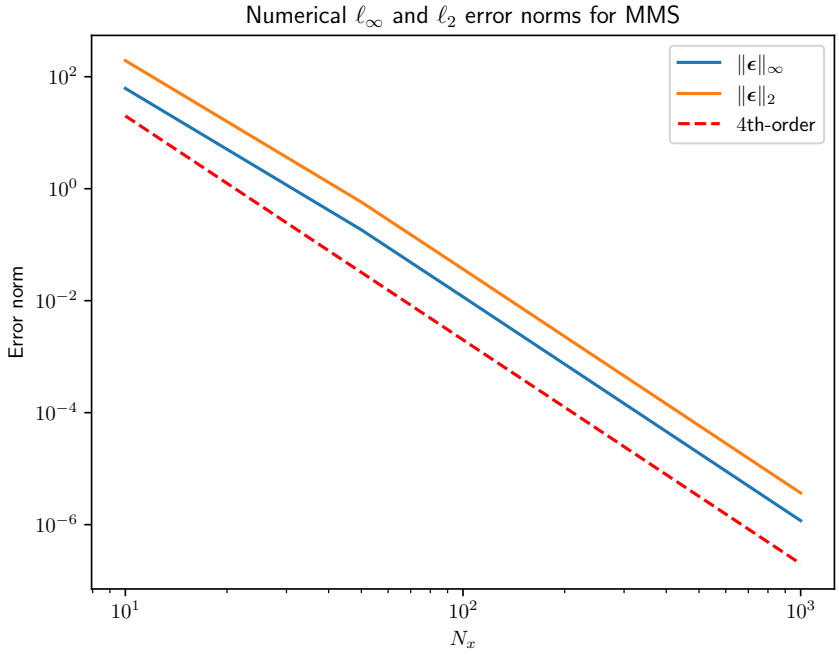
Solution: In order to make sure our previously developed code works properly, we verify it using the method of manufactured solutions. For this, we prescribe a manufactured solution that is L-periodic and smooth and analytically solve for the required source terms that we add to the original ODE to acquire the modified equations

$$\frac{\partial \mathbf{q}}{\partial t} = -\frac{\partial \mathbf{f}(\mathbf{q})}{\partial x} + \begin{bmatrix} S_0 \\ S_1 \\ S_2 \end{bmatrix}. \tag{31}$$

The manufactured solution is then a solution to the modified equations, which we can evaluate numerically to check if the order of accuracy implemented into the RHS evaluation holds true. For the manufactured solutions, we chose

$$\rho = a_1 + a_2 \sin(kx)$$
$$u = a_3 + a_4 \sin(kx)$$
$$p = a_5 + a_6 \sin(kx)$$

with $k = 2\pi/L$. We solve for the resulting source terms $S_i(x)$ symbolically and implement into the existing function that contains and evaluates the RHS of the ODE. Since the manufactured solution is steady, the RHS for given solution should be equal to zero, aside from numerical errors (i.e. finite difference approximation), which we know is of 4^{th} order accuracy. Conducting a grid convergence study, we see that the resulting error follows the expected order of accuracy for the L_2- and L_∞-norm.



Solution: Full codes:

Python (0-based indexing)

```python
# %% [markdown]
# Problem 4: Eigendecomposition of the flux Jacobian

# %%
import sympy as sym
from IPython.display import display, Latex

# Declare symbolic variables
rho, u, c, g, H = sym.symbols('rho u c gamma H')

# Construct Jacobian matrix with symbolic variables
M = sym.Matrix([[0, 1, 0],[(g-3)*u*u*sym.Rational(1,2), (3-g)*u, g-1], [(g-2)*u*u*u*sym.Rational(1,2)-c*c*u/(g-1), c*c/(g-1)-(g-sym.Rational(3,2))*u*u, g*u]])

# Compute and print eigenvalues and eigenvectors
EV = M.eigenvects()
for i in range(3):
  # Extract eigenvalue and eigenvector
  Eval = EV[i][0]
  Evec = EV[i][2][0]
  # Normalize eigenvector by first component
  Evec /= Evec[0]
  # Substitute c^2 by (gamma-1)*(H-u^2/2)
  Evec = Evec.subs(c**2, (g-1)*(H-u**2*sym.Rational(1,2)))
  # Simplify
  Evec = sym.simplify(Evec)
  # Print eigenvalue and eigenvector
  display(Latex('$\Omega_{} = {}$'.format(i,sym.latex(Eval))))
  display(Latex('$\mathbf{}X{}_{} = {}$'.format('{','}',i,sym.latex(Evec))))

# %% [markdown]
# Problem 6: Modified wavenumber for the 4th-order upwind scheme

# %%
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import cmath
plt.rcParams['text.usetex'] = True
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300
plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{amssymb}')

# We only plot the first N/2+1 eigenvalues, corresponding to k*dx in [0,pi]
Nx = 150
halfNx = int(0.5*Nx+1)
kndx = np.zeros(halfNx)

# Initialize abscissae
for n in range(halfNx):
    kndx[n] = 2*np.pi*n/Nx

# Exact modified wavenumbers
knmoddx_exact_upwind4_plus  = (21*np.sin(kndx)-6*np.sin(2*kndx)+np.sin(3*kndx))/12 - (-15*np.cos(kndx)+6*np.cos(2*kndx)-np.cos(3*kndx)+10) * 1j/12
knmoddx_exact_upwind4_minus  = (21*np.sin(kndx)-6*np.sin(2*kndx)+np.sin(3*kndx))/12 - (15*np.cos(kndx)-6*np.cos(2*kndx)+np.cos(3*kndx)-10) * 1j/12

# Plot real part
fig, ax = plt.subplots(1, 1, figsize=(5, 4))
ax.plot(kndx, kndx, 'k', label=r'Exact')
ax.plot(kndx, knmoddx_exact_upwind4_plus.real, color='red', label=r'Upwind 4 Plus - Analytical')
```

```
ax.plot(kndx, knmoddx_exact_upwind4_minus.real, color='blue', linestyle='dashed', label=r'Upwind 4 Minus - Analytical')
plt.ylabel(r'$\mathbb{R}\mathrm{e}(\kappa^\star \Delta x)$');plt.xlabel(r'$\kappa\Delta x$')
plt.legend()
tick_labels = [r'$0$',r'$\pi/2$',r'$\pi$']
ticks = [0,np.pi/2,np.pi]
ax.set_xticks(ticks); ax.set_yticks(ticks)
ax.set_xticklabels(tick_labels); ax.set_yticklabels(tick_labels)
plt.savefig('modwavenumber_real.pdf')
plt.show()

# Plot imaginary part
fig, ax = plt.subplots(1, 1, figsize=(5, 4))
ax.plot(kndx, 0*kndx, 'k', label=r'Exact')
ax.plot(kndx, knmoddx_exact_upwind4_plus.imag, color='red', label=r'Upwind 4 Plus - Analytical')
ax.plot(kndx, knmoddx_exact_upwind4_minus.imag, color='blue', linestyle='dashed', label=r'Upwind 4 Minus - Analytical')
plt.ylabel(r'$\mathbb{I}\mathrm{m}(\kappa^\star \Delta x)$');plt.xlabel(r'$\kappa\Delta x$')
plt.legend()
ytick_labels = [r'$0$',r'$-1$',r'$-2$']
yticks = [0,-1,-2]
ax.set_xticks(ticks); ax.set_yticks(yticks)
ax.set_xticklabels(tick_labels); ax.set_yticklabels(ytick_labels)
plt.savefig('modwavenumber_imag.pdf')
plt.show()

# %% [markdown]
# Problem 7: Integrator for the 1D Euler equation using flux-splitted finite differences:

# %%
from scipy.linalg import circulant
from scipy.integrate import solve_ivp

# Operator from HW1
def D_operator_periodic(N,L,R,a):
    first_row = np.zeros(N); first_row[0:L+R+1] = a; first_row = np.roll(first_row,-L)
    return np.array(circulant(first_row)).transpose()

# Functor for the 1D Euler equation
def Euler1D(t,Q,Dp,Dm):
    # Ratio of specific heats
    gamma = 1.4
    # Number of computational nodes (obtained from the size of D)
    Nx = int(len(Dp))
    # Extract conserved variables from the vector Q
    rho = Q[:Nx]
    rhoU = Q[Nx:2*Nx]
    rhoE = Q[2*Nx:]
    # Reconstruct velocity from conserved variables
    U = rhoU/rho
    U_sq = U*U
    # Reconstruct pressure from conserved variables using the EOS
    p = (gamma-1)*(rhoE-np.multiply(rhoU,U)/2)
    # Compute speed of sound
    c_sq = gamma*p/rho
    c = np.sqrt(c_sq)
    # Compute total specific enthalpy
    H = c_sq/(gamma-1)+U_sq/2
    # Compute spatial derivatives of conserved variables with +/- FD biased schemes
    drhodxp = Dp@rho; drhodxm = Dm@rho
    drhoUdxp = Dp@rhoU; drhoUdxm = Dm@rhoU
    drhoEdxp = Dp@rhoE; drhoEdxm = Dm@rhoE
    # Initialize Qdot = F(t,Q) to zero
```

```
121     F = np.zeros(len(Q))
122     # Fill in Qdot = F(t,Q) at each discrete grid point
123     for i in range(Nx):
124         # Construct local diagonal matrix of eigenvalues
125         Omega = np.zeros((3,3)); Omega[0][0]=U[i]; Omega[1][1]=U[i]-c[i]; Omega[2][2]=U[i
                ]+c[i]
126         # Split this matrix into positive and negative eigenvalues
127         Omegap = (Omega + np.abs(Omega))/2; Omegam = Omega-Omegap
128         # Compute the corresponding matrix of eigenvectors and its inverse
129         X = np.array([[1,U[i],U_sq[i]/2],[1,U[i]-c[i],H[i]-U[i]*c[i]],[1,U[i]+c[i],H[i]+U[
                i]*c[i]]]).transpose()
130         Xinv = np.linalg.inv(X)
131         # Construct flux-splitted matrices M+/- = X@(Omega+/-)@X^{-1}
132         Mp = X@Omegap@Xinv
133         Mm = X@Omegam@Xinv
134         # Compute local derivatives and F(t,Q) for the ith cell
135         dQdxp = np.array([drhodxp[i],drhoUdxp[i],drhoEdxp[i]])
136         dQdxm = np.array([drhodxm[i],drhoUdxm[i],drhoEdxm[i]])
137         F_local = -Mp@dQdxp - Mm@dQdxm
138         # Update the global vector that contains all F(t,Q)
139         for j in range(3): F[j*Nx+i] = F_local[j]
140     return F
141 
142 # General integrator function
143 def Integrator(periodic,operator,problem,L,T,Nx,Nt,U0):
144     # Initialize spatial domain
145     x  = np.linspace(0, L, Nx, endpoint=(not periodic))
146     dx = x[1] - x[0]
147 
148     # Initialize temporal domain
149     t  = np.linspace(0, T, Nt, endpoint=True)
150 
151     # Construct spatial matrix operator
152     match (operator,periodic):
153         case ('UpwindOrder4FirstDeriv',True):       # Periodic 3rd-order upwind
154             Dp = D_operator_periodic(Nx,3,1,[-1,6,-18,10,3]/(12*dx))
155             Dm = D_operator_periodic(Nx,1,3,[-3,-10,18,-6,1]/(12*dx))
156         case _:
157             raise Exception("The %s operator '%s' is not yet implement!" % ('periodic' if
                    periodic else 'non-periodic', operator))
158 
159     # Solve and return solutions!
160     match problem:
161         case 'Euler1D':
162             # Solve initial value problem; see documentation at:
163             # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.
                    solve_ivp.html
164             sol = solve_ivp(Euler1D, [0, T], U0, args=(Dp,Dm), t_eval=t, rtol=1.0e-9, atol
                    =1.0e-9, first_step=1e-6)
165             # Transpose solution vector so U has the format (Nt x Nx)
166             U = sol.y.transpose()
167             # Return outputs
168             return t, U
169         case _:
170             raise Exception("The case '%s' is not yet implement!" % problem)
171 
172 # %% [markdown]
173 # Problem 8: Solution of an example of 1D Euler periodic problem
174 
175 # %%
176 # Initialize space and temporal domains
177 L = 10; T = 0.02
178 Nx = 150; Nt = 6
179 x = np.linspace(0, L, Nx, endpoint=False)
180 dx = x[1] - x[0]
181 
```

```
# Intialize conserved variables
Q0 = np.zeros(3*Nx)
gamma = 1.4
for i in range(Nx):
    rho = 1.225
    U = 100
    p = 101325*(1+0.1*np.exp(-10*(x[i]-5.0)**2))
    Q0[i] = rho
    Q0[Nx+i] = rho*U
    Q0[2*Nx+i] = p/(gamma-1) + rho*U**2/2

# Solve problem!
t, Q = Integrator(True, 'UpwindOrder4FirstDeriv', 'Euler1D',L, T, Nx, Nt, Q0)

# Extract conserved variables from the vector Q, and reconstruct velocity and pressure
rho = Q[:,0:Nx]
rhoU = Q[:,Nx:2*Nx]
rhoE = Q[:,2*Nx:3*Nx]
U = np.divide(rhoU,rho)
p = (gamma-1)*(rhoE-rhoU*rhoU/rho/2)

# %%
# Plot solution
for j in range(Nt):
    fig, (ax_rho, ax_u, ax_p) = plt.subplots(1, 3, sharex=True, figsize=(10, 2))
    ax_rho.plot(x,rho[j])
    ax_u.plot(x,U[j],'r')
    ax_p.plot(x,p[j]*1e-3,'k')
    ax_rho.set_title(r'$\rho(x,%.3f)$' % t[j]); ax_rho.set_ylim(1.13,1.3); ax_rho.
        set_xlabel(r'$x$')
    ax_u.set_title(r'$u(x,%.3f)$' % t[j]); ax_u.set_ylim(85,115); ax_u.set_xlabel(r'$x$')
    ax_p.set_title(r'$p(x,%.3f) \times 10^{-3}$' % t[j]); ax_p.set_ylim(100,112); ax_p.
        set_xlabel(r'$x$')
    plt.savefig('euler1d_%i.pdf' % j)
    plt.show()

# %% [markdown]
# Problem 9: Verification with the method of manufactured solutions (MMS)

# %%
x, rho, u, p, g, k = sym.symbols('x rho u p g k')
a1, a2, a3, a4, a5, a6 = sym.symbols('a1 a2 a3 a4 a5 a6')

# Initialize prescribed variables symoblically
rho = a1+a2*sym.sin(k*x)
u   = a3+a4*sym.sin(k*x)
p   = a5+a6*sym.sin(k*x)

# Declare flux functions
f0 = rho*u
f1 = rho*u*u+p
f2 = (p/(g-1)+rho*u*u/2+p)*u

# Calculate source terms for the MMS (symbolically)
S0 = sym.diff(f0,x)
S1 = sym.diff(f1,x)
S2 = sym.diff(f2,x)
print('S0 = ', S0)
print('S1 = ', S1)
print('S2 = ', S2)

# Create numerical function that returns source terms from the coefficients and positions
MMS = sym.lambdify([a1,a2,a3,a4,a5,a6,g,k,x], [S0,S1,S2], 'numpy')

# %%
# Initialize mesh resolutions
```

```
L = 1; Nxs = [10,50,100,500,1000]
k = 2*np.pi/L

# Initialize MMS coefficients
a1 = 1
a2 = 0.5
a3 = -1
a4 = 4
a5 = 42
a6 = 0.7

L2Error = []
LinfError = []
# Loop over all considered mesh resolutions
for n in range(len(Nxs)):
    # Create grid
    Nx = Nxs[n]
    x = np.linspace(0, L, Nx, endpoint=False)
    dx = x[1] - x[0]

    # Intialize conserved variables according to MMS function
    Q0 = np.zeros(3*Nx)
    gamma = 1.4
    rho = a1+a2*np.sin(k*x)
    u   = a3+a4*np.sin(k*x)
    p   = a5+a6*np.sin(k*x)
    Q0[:Nx] = rho
    Q0[Nx:2*Nx] = rho*u
    Q0[2*Nx:] = p/(gamma-1)+rho*u*u/2

    # Compute dQ/dt = F(t,Q) using the 1D Euler functor
    Dp = D_operator_periodic(Nx,3,1,[-1,6,-18,10,3]/(12*dx))
    Dm = D_operator_periodic(Nx,1,3,[-3,-10,18,-6,1]/(12*dx))
    F = Euler1D(0,Q0,Dp,Dm)

    # Compute error (i.e., the difference between -F and the MMS source terms)
    Error = F + np.reshape(MMS(a1,a2,a3,a4,a5,a6,gamma,k,x),3*Nx)
    L2Error.append(np.max(np.abs(Error)))
    LinfError.append(np.sqrt(np.sum(Error**2)/(3*Nx)))

# Plot numerical error norms
plt.title(r'Numerical $\ell_\infty$ and $\ell_2$ error norms for MMS')
plt.xlabel(r'$N_x$');plt.ylabel(r'Error norm')
plt.loglog(Nxs,LinfError,label=r'$\|\boldsymbol{\epsilon}\|_\infty$')
plt.loglog(Nxs,L2Error,label=r'$\|\boldsymbol{\epsilon}\|_2$')
plt.loglog([1e1,1e3],[2e1,2e-7],'r--',label=r'$4$th-order')
plt.legend()
plt.savefig('MMS.pdf')
plt.show()
```

Matlab (1-based indexing)

```
function [ D ] = D1_operator( n,dx,R,L,a )
%computes a finite difference operator
%   n = number of grid points
%   dx = step size
%   [R,L]  = right/left bound of stencil
%   a   = stencils

% initialize
D = zeros(n+1);
% fill interior domain
for i=1+L:n+1-R
    D(i,i-L:i+R) = a;
```

```
13  end
14  % fill left boundary
15  for i=1:L
16      D(i,1:i+R)          = a(L-i+2:end);
17      D(i,end-L+i:end)    = a(1:L-i+1);
18  end
19  % fill right boundary
20  for i=n+2-R:n+1
21      D(i,end-L-1+(i-n):end)  = a(1:L+(n+2-i));
22      D(i,1:(i-n+R-1))        = a(L+(n+3-i):end);
23  end
24  D = 1/dx*D;
25  end
```

```
1   function qdot = Euler1D(t,q,dx,Dp,Dm)
2   clc; t
3
4   % ideal gas
5   gam     = 1.4;
6   %
7   Nx      = length(Dp);
8
9   % extract current state in primitive variables
10  rho     = q(1:Nx);
11  rhou    = q(Nx+1:2*Nx);
12  rhoE    = q(2*Nx+1:3*Nx);
13  u       = rhou./rho;
14  p       = (gam-1)*(rhoE-0.5*rho.*u.^2);
15
16  % compute Lambda
17  a_sq    = gam*p./rho;
18  for i=1:Nx
19      if (a_sq(i) < 0)
20          %error('contains negative element');
21      end
22  end
23  a       = sqrt(gam*p./rho);
24  Lambda1 = (u-a);
25  Lambda2 = u;
26  Lambda3 = u+a;
27  Os      = zeros(Nx,Nx);
28  Lambda  = [diag(Lambda1),Os,Os;Os,diag(Lambda2),Os;Os,Os,diag(Lambda3)];
29  Lambdap = 0.5*( Lambda + abs(Lambda) );
30  Lambdam = Lambda-Lambdap;
31
32  % compute matrix containing eigenvectors
33  H       = 0.5*u.^2+a.^2/(gam-1);
34
35  % eigenvectors
36  V11     = eye(Nx);
37  V21     = diag((u-a));
38  V31     = diag(H-u.*a);
39  V12     = eye(Nx);
40  V22     = diag(u);
41  V32     = diag(0.5*u.^2);
42  V13     = eye(Nx);
43  V23     = diag(u+a);
44  V33     = diag(H+u.*a);
45  V       = [V11,V12,V13;V21,V22,V23;V31,V32,V33];
46
47  % compute A+ and A- matrices
48  Vinv    = inv(V);
49  Ap      = V*Lambdap*Vinv;
50  Am      = V*Lambdam*Vinv;
51
52  % compute global differention operator
```

```
Dpmat   = [Dp,Os,Os;Os,Dp,Os;Os,Os,Dp];
Dmmat   = [Dm,Os,Os;Os,Dm,Os;Os,Os,Dm];

% compute RHS
qdot    = -( Ap*(Dpmat*q) + Am*(Dmmat*q));
return
```

```
function [t,q] = integrator(x,T,Nt,q0)
%
% Inputs
%
% x :: grid point vector
% T :: length of temporal domain, t runs from 0 to T
% Nt :: number of points to use in t (for reporting solution, ode45 chooses dt internally)
% t :: time vector
% q :: solution vector

% spatial stepsize
dx = x(2)-x(1);

% temporal domain
dt = T/(Nt-1);
t = dt*linspace(0,Nt-1,Nt);

% spatial operators
ap      = [-1,6,-18,10,3]/12;
am      = [-3,-10,18,-6,1]/12;
[ Dp ]  = D1_operator( length(x)-1,dx,1,3,ap );
[ Dm ]  = D1_operator( length(x)-1,dx,3,1,am );

% call ode45
[t,q] = ode45(@(t,y) Euler1D(t,y,dx,Dp,Dm), t, q0);
end
```

```
clear all; clc; close all;

syms gam u a

A = [0,1,0;...
     -1/2*(3-gam)*u^2,(3-gam)*u,(gam-1);...
     u*(1/2*(gam-2)*u^2-a^2/(gam-1)),a^2/(gam-1)-(gam-3/2)*u^2,gam*u];

[vecs,vals] = eig(A);

res1 = A*vecs(:,1)-vals(1,1)*vecs(:,1)
res2 = A*vecs(:,2)-vals(2,2)*vecs(:,2)
res3 = A*vecs(:,3)-vals(3,3)*vecs(:,3)
```

```
clear all; close all; clc

syms dx a1 a2 a3 a4 a5

%% upwind
A = [1,1,1,1,1;...
     -dx,0,dx,2*dx,3*dx;...
     dx^2,0,dx^2,(2*dx)^2,(3*dx)^2;...
     -dx^3,0,dx^3,(2*dx)^3,(3*dx)^3;...
     dx^4,0,dx^4,(2*dx)^4,(3*dx)^4];
b = [0;1;0;0;0];

x = inv(A)*b


%% downwind
A = [1,1,1,1,1;...
     dx,0,-dx,-2*dx,-3*dx;...
```

```
      dx^2,0,dx^2,(2*dx)^2,(3*dx)^2;...
      dx^3,0,-dx^3,-(2*dx)^3,-(3*dx)^3;...
      dx^4,0,dx^4,(2*dx)^4,(3*dx)^4];
b = [0;1;0;0;0];

x = inv(A)*b
```

```
clear all; close all; clc
n = 150;

for k=1:n
    dxkm(k)     = 2*pi*(k-1)/(n-1);
end

modWaveNumAn1 = 1/(sqrt(-1)*12)*( ( 15*cos(dxkm)-6*cos(2*dxkm)+cos(3*dxkm)-10 ) ) ...
                +1/(sqrt(-1)*12)*( sqrt(-1)*( 21*sin(dxkm)-6*sin(2*dxkm)+sin(3*dxkm) ) );

modWaveNumAn2 = 1/(sqrt(-1)*12)*( -( 15*cos(dxkm)-6*cos(2*dxkm)+cos(3*dxkm)-10 ) ) ...
                +1/(sqrt(-1)*12)*( sqrt(-1)*( 21*sin(dxkm)-6*sin(2*dxkm)+sin(3*dxkm) ) );

% plot modified wavenumber
line   = 3;
markerssize = 8;
gcafont     = 20;
labelsize   = 24;
legendsize  = 24;

figure(3), clf; hold on
subplot(1,2,1); hold on
plot(dxkm,real(modWaveNumAn1),'k-','LineWidth',3,'LineWidth',line)
plot(dxkm,real(modWaveNumAn2),'--','LineWidth',3,'LineWidth',line)
plot([0,pi],[0,pi],'r-','LineWidth',2)
xlim([0,pi]); ylim([0,pi])
set(gca,'FontSize',gcafont)
xlabel('$k_m \Delta x$','Interpreter','Latex','FontSize',labelsize)
ylabel('$\Re(\tilde{k}_m \Delta x)$','Interpreter','Latex','FontSize',labelsize)
leg=legend('$D^+$','$D^-$','exact');
set(leg,'Interpreter','Latex','FontSize',legendsize,'Location','NorthWest')

subplot(1,2,2); hold on
plot(dxkm,-imag(modWaveNumAn1),'k-','LineWidth',3,'LineWidth',line)
plot(dxkm,-imag(modWaveNumAn2),'--','LineWidth',3,'LineWidth',line)
plot([0,pi],[0,0],'r-','LineWidth',2)
xlim([0,pi]);% ylim([0,pi])
set(gca,'FontSize',gcafont)
xlabel('$k_m \Delta x$','Interpreter','Latex','FontSize',labelsize)
ylabel('$\Im(\tilde{k}_m \Delta x)$','Interpreter','Latex','FontSize',labelsize)
leg=legend('$D^+$','$D^-$','exact');
set(leg,'Interpreter','Latex','FontSize',legendsize,'Location','NorthWest')

set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 100 1000 300])
print(gcf,'midterm_P6','-djpeg90')
```

```
clear all; clc; close all;

Nxs     = [10,50,100,500,1000];
L       = 1;
gam     = 1.4;
pi2L    = 2*pi/L;
t       = 1;
ap      = [-1,6,-18,10,3]/12;
am      = [-3,-10,18,-6,1]/12;

%% source terms
a(1) = 1;
```

```
a(2) = 0.5;
a(3) = -1;
a(4) = 4;
a(5) = 0.7;
a(6) = 42;

for ii=1:length(Nxs)
    Nx      = Nxs(ii);

    %% compute x-domain
    x       = linspace(0,L,Nx+1);
    x       = x(1:end-1);
    dx = x(2)-x(1);

    %% compute initial condition
    rho0    = (a(1) + a(2)*sin(pi2L*x))';
    u0      = (a(3) + a(4)*sin(pi2L*x))';
    rhou0   = u0.*rho0;
    p0      = (a(5) + a(6)*sin(pi2L*x))';
    rhoE0   = p0/(gam-1)+0.5*rhou0.^2./rho0;
    q0      = [rho0;rhou0;rhoE0];

    %% compute RHS with source terms
    [ Dp ]      = D1_operator( length(x)-1,dx,1,3,ap );
    [ Dm ]      = D1_operator( length(x)-1,dx,3,1,am );
    qdot        = Euler1D(t,q0,dx,Dp,Dm);
    [S1,S2,S3]  = sourceterms(a,t,x,L,gam);

    %% compute error and norms
    err         = qdot +[S1;S2;S3];
    L2norm(ii)     = norm(err);
    Linfnorm(ii)     = max(abs(err));
end

%% plot err norms
figure(1), clf;
loglog(Nxs,L2norm,'LineWidth',2); hold on
loglog(Nxs,Linfnorm,'k-','LineWidth',2);
loglog([1e1,1e3],[1e3,1e-4],'r--','LineWidth',2)
loglog([1e1,1e3],[1e2,1e-6],'r:','LineWidth',2)
xlabel('$N_x$','Interpreter','Latex','FontSize',30)
set(gca,'FontSize',20)
leg=legend('L$_2$-norm','L$_{\infty}$-norm','$3.5^{th}$-order','$4^{th}$-order');
set(leg,'Interpreter','Latex','FontSize',24)
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 100 800 300])
print(gcf,'midterm_P8_errnorms','-djpeg90')
```

```
clear all; clc; close all;

legfont     = 25;
axisfont    = 18;
labelfont   = 30;

%% input parameter
T   = 0.02;
Nt  = 6;
L   = 10;
mu  = 5;
sigma = 0.1;
Nx  = 150;

%% compute x-domain
x   = linspace(0,L,Nx+1);
x   = x(1:end-1);

```

```
%% compute initial condition
gam     = 1.4;
rho0    = 1.225*ones(Nx,1);
rhou0   = 1.225*100*ones(Nx,1);
pinf    = 101325;
p0      = pinf*(1+0.1*exp(-(x-mu).^2/sigma))';
rhoE0   = p0/(gam-1)+0.5*rhou0.^2./rho0;
q0      = [rho0;rhou0;rhoE0];

%% compute unsteady solution and eigenvalues of differentiation operators
[t,q] = integrator(x,T,Nt,q0);

%% plot
figure(1), clf; set(gcf,'Position',[100 100 1000 400])
for tt=1:length(t)
    clf
    rho     = q(tt,1:Nx);
    rhou    = q(tt,Nx+1:2*Nx);
    rhoE    = q(tt,2*Nx+1:3*Nx);

    u       = rhou./rho;
    p       = (gam-1)*(rhoE-0.5*rho.*u.^2);

    a(1)=subplot(1,3,1);
    plot(x,rho0,'r--','LineWidth',2); hold on
    plot(x,rho,'k-','LineWidth',2);
    leg1=legend('IC','$\rho$');
    ylim([1.13,1.3])
    a(2)=subplot(1,3,2);
    plot(x,rhou0./rho0,'r--','LineWidth',2); hold on
    plot(x,u,'k-','LineWidth',2);
    leg2=legend('IC','$u$');
    ylim([85,115])
    title(['t=',num2str(t(tt),'%0.3f')],'FontSize',16)
    a(3)=subplot(1,3,3);
    plot(x,p0,'r--','LineWidth',2); hold on
    plot(x,p,'k-','LineWidth',2);
    leg3=legend('IC','$p$');
    ylim([100,112]*1e3)

    set(a,'FontSize',axisfont)
    xlabel(a(1),'$x$','Interpreter','Latex','FontSize',labelfont)
    xlabel(a(2),'$x$','Interpreter','Latex','FontSize',labelfont)
    xlabel(a(3),'$x$','Interpreter','Latex','FontSize',labelfont)
    set(leg1,'Interpreter','Latex','FontSize',legfont)
    set(leg2,'Interpreter','Latex','FontSize',legfont)
    set(leg3,'Interpreter','Latex','FontSize',legfont)

    set(gcf,'PaperPositionMode','Auto')
    set(gcf,'Position',[100 100 1000 300])
    fname   = ['midterm_P9_soln_tt0',num2str(tt),'.jpg'];
    print(gcf,fname,'-djpeg90')
    pause(0.01)
end
```

```
clear all; close all; clc

syms x pi2L t a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 gam

% prescribed variables
rho = a1 + a2*sin(pi2L*x);
u   = a3 + a4*sin(pi2L*x);
p   = a5 + a6*sin(pi2L*x);

% derived variables
q1  = rho;
```

```
q2  = rho*u;
q3  = p/(gam-1)+0.5*rho*u^2;
rhoE= q3;

% fluxes
f1  = rho*u;
f2  = (gam-1)*rhoE + 0.5*(3-gam)*rho*u^2;
f3  = gam*u*rhoE-0.5*(gam-1)*rho*u^3;

% source terms
S1  = diff(f1,x)
S2  = diff(f2,x)
S3  = diff(f3,x)
```

```
function [ S1,S2,S3 ] = sourceterms(a,t,x,L,gam)

pi2L = 2*pi/L;
a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);
a5 = a(5);
a6 = a(6);

% new source terms
S1 = a4.*pi2L.*cos(pi2L*x).*(a1 + a2*sin(pi2L*x)) + a2*pi2L*cos(pi2L*x).*(a3 + a4*sin(pi2L
    *x));
S2 = (gam - 1).*((a6.*pi2L.*cos(pi2L.*x))/(gam - 1) + (a2.*pi2L.*cos(pi2L.*x).*(a3 + a4.*
    sin(pi2L.*x)).^2)/2 + 2.*a4.*pi2L.*cos(pi2L.*x).*(a3 + a4.*sin(pi2L.*x)).*(a1/2 + (a2
    .*sin(pi2L.*x))/2)) - a2.*pi2L.*cos(pi2L.*x).*(gam/2 - 3/2).*(a3 + a4.*sin(pi2L.*x))
    .^2 - 2.*a4.*pi2L.*cos(pi2L.*x).*(gam/2 - 3/2).*(a1 + a2.*sin(pi2L.*x)).*(a3 + a4.*sin
    (pi2L.*x));
S3 = gam.*(a3 + a4.*sin(pi2L.*x)).*((a6.*pi2L.*cos(pi2L.*x))/(gam - 1) + (a2.*pi2L.*cos(
    pi2L.*x).*(a3 + a4.*sin(pi2L.*x)).^2)/2 + 2.*a4.*pi2L.*cos(pi2L.*x).*(a3 + a4.*sin(
    pi2L.*x)).*(a1/2 + (a2.*sin(pi2L.*x))/2)) - a2.*pi2L.*cos(pi2L.*x).*(gam/2 - 1/2).*(a3
     + a4.*sin(pi2L.*x)).^3 + a4.*gam.*pi2L.*cos(pi2L.*x).*((a3 + a4.*sin(pi2L.*x)).^2.*(
    a1/2 + (a2.*sin(pi2L.*x))/2) + (a5 + a6.*sin(pi2L.*x))/(gam - 1)) - 3.*a4.*pi2L.*cos(
    pi2L.*x).*(gam/2 - 1/2).*(a1 + a2.*sin(pi2L.*x)).*(a3 + a4.*sin(pi2L.*x)).^2;

S1  = S1';
S2  = S2';
S3  = S3';
end
```

Submission guidelines · Instructions on how to prepare and submit your report are available on the course's Canvas page at https://canvas.illinois.edu/courses/43781/assignments/syllabus