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

Homework #4

Due date: April 11, 2024

In this homework, you will solve the inviscid Burgers equation using finite volumes and the Godunov method. The inviscid Burgers equation is given as

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(f(u)) = 0 , \quad \text{with } f(u) = \frac{u^2}{2} . \tag{1}$$

Problem 1 · Suppose an initial condition $u(x, t = 0) = u_0(x)$ that is smooth, i.e., differentiable everywhere.

Q1 → Show that the exact solution to Burgers' equation with this initial condition is given implicitly as

$$u(x, t) = u_0(x - tu(x, t)) . \tag{2}$$

Note: Since you will need to use the differential form of Burgers' equation to get to this result, this expression is only valid until the solution forms its first shockwave (i.e., until it is not differentiable everywhere anymore).

Solution: For Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{3}$$

with the initial condition

$$u(x, t = 0) = u_0(x) \tag{4}$$

we know the exact solution to be

$$u(x, t) = u_0(x - ut) \tag{5}$$

so we can rewrite the derivatives as follows:

$$\frac{\partial u}{\partial t} = u_0'\left(-\frac{\partial u}{\partial t}t - u\right)$$
$$\frac{\partial u}{\partial x} = u_0'\left(1 - \frac{\partial u}{\partial x}t\right)$$

where the prime denotes the derivative with respective to the argument $x - ut$. Plugging these into Burgers' Equation, we get

$$\begin{aligned} u_0'\left(-\frac{\partial u}{\partial t}t - u\right) + uu_0'\left(1 - \frac{\partial u}{\partial x}t\right) &= 0 \\ u_0'\left(-\frac{\partial u}{\partial t}t - u + u - u\frac{\partial u}{\partial x}t\right) &= 0 \\ -tu_0' \underbrace{\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x}\right)}_{\text{Burgers' Equation } = 0} &= 0 \quad \text{q.e.d.} \end{aligned}$$

Problem 2 · To solve Burgers' equation using finite volumes, you will employ Godunov's method. This requires knowing the solutions to the Riemann problem corresponding to the initial conditions

$$u(x,0) = \begin{cases} u_L & \text{if } x \leq 0 \\ u_R & \text{if } x > 0 \end{cases} \tag{6}$$

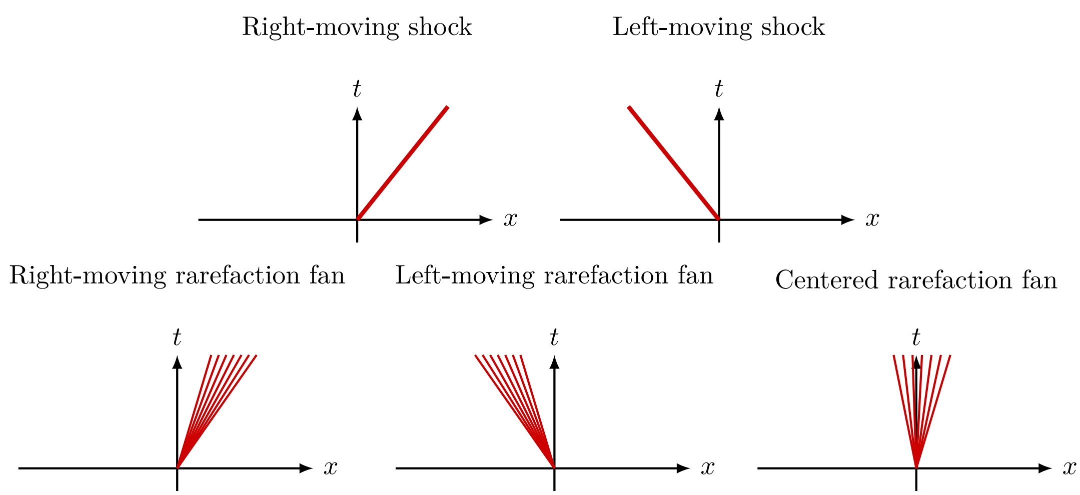
Q2.1 → Derived the five fundamental solutions to this Riemann problem and express them in terms of u_L, u_R, and the similarity variable $\xi = x/t$.

Solution: The local Riemann problem has five possible solutions, depending on the quantities u_L and u_R, which denote the cell average velocity on the left and right side of the interface or the discontinuity respectively: left- and right-moving shock and left-moving, right-moving, and centered expansion fan (see Figure below). For the conditions of the two distinctively different solutions of shocks and expansion fans, we can state that when $u_L > u_R$

$$u^\star(\xi) = \begin{cases} u_L & \text{if} \quad S \geq \xi \\ u_R & \text{if} \quad S \leq \xi \end{cases} \tag{7}$$

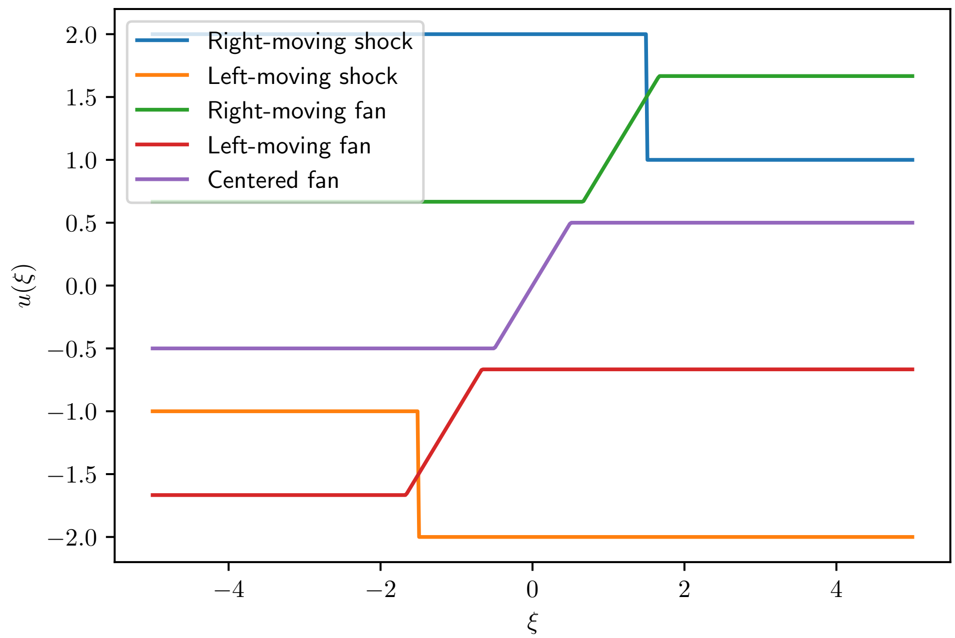
with $\xi = \frac{x}{t}$ and the shock speed $S = \frac{1}{2}(u_L + u_R)$. For expansion fans under the condition $u_L \leq u_R$, we can state

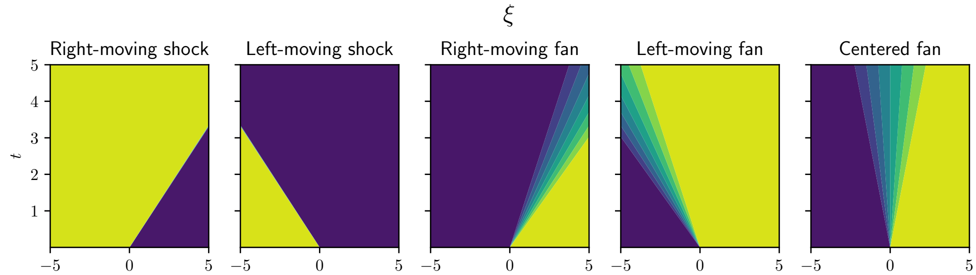
$$u^\star(\xi) = \begin{cases} u_L & \text{if} \quad \xi \leq u_L \\ \xi & \text{if} \quad u_L < \xi < u_R \\ u_R & \text{if} \quad \xi \geq u_R \end{cases} \tag{8}$$



Q2.2 → Develop a computational *Riemann solver* that takes u_L, u_R, and ξ as inputs and returns the corresponding solution to the Riemann problem for the similarity variable ξ. Verify that each case is correctly implemented by testing relevant combinations of u_L and u_R.

Solution: A function has been developed that solves the local Riemann problem. For verification, a set of parameters has been chosen such that we cover all 5 possible flow scenarios: left-moving and right-moving shock as well as left-moving, right-moving, and centered expansion fan. The solution is a function of the similarity variable $\eta = \frac{x}{t}$, which can be seen in the first figure below. Alternatively, we can convert back onto the 2-dimensional x-t plane to make it more visually accessible, as can be seen in the second figure below. As can be seen in these figures, the Riemann solver adequately solves the Riemann problem for all five flow scenarios and will be subsequently used as part of a finite volume solver for Burgers' Equation.





Problem 3 · When using finite volumes, the discrete variables under consideration are the cell averages of the solution $u(x,t)$. To be rigorous, initializing our simulation thus requires to calculate the cell averages of the initial condition $u(x,0)$.

Q3.1 → Develop a computational function that returns the average of any function $\mathcal{F}(x)$ on an interval $[a,b]$ using the trapezoidal integration rule:

$$\frac{1}{b-a}\int_a^b \mathcal{F}(x)\ \mathrm{d}x \simeq \frac{1}{2K}\sum_{k=0}^{K-1}\left[\mathcal{F}\left(a+(b-a)\frac{k}{K}\right)+\mathcal{F}\left(a+(b-a)\frac{k+1}{K}\right)\right]\ , \tag{9}$$

The inputs of this function should be:

- The bounds a and b.
- The function $\mathcal{F}$.
- The number of integration points K.

Its only output should be:

The (approximated) average of $\mathcal{F}(x)$ on $[a, b]$.

Q3.2 → Verify your code by computing the cell-averages of the function

$$\mathcal{F}(x) = \sin(10x) \tag{10}$$

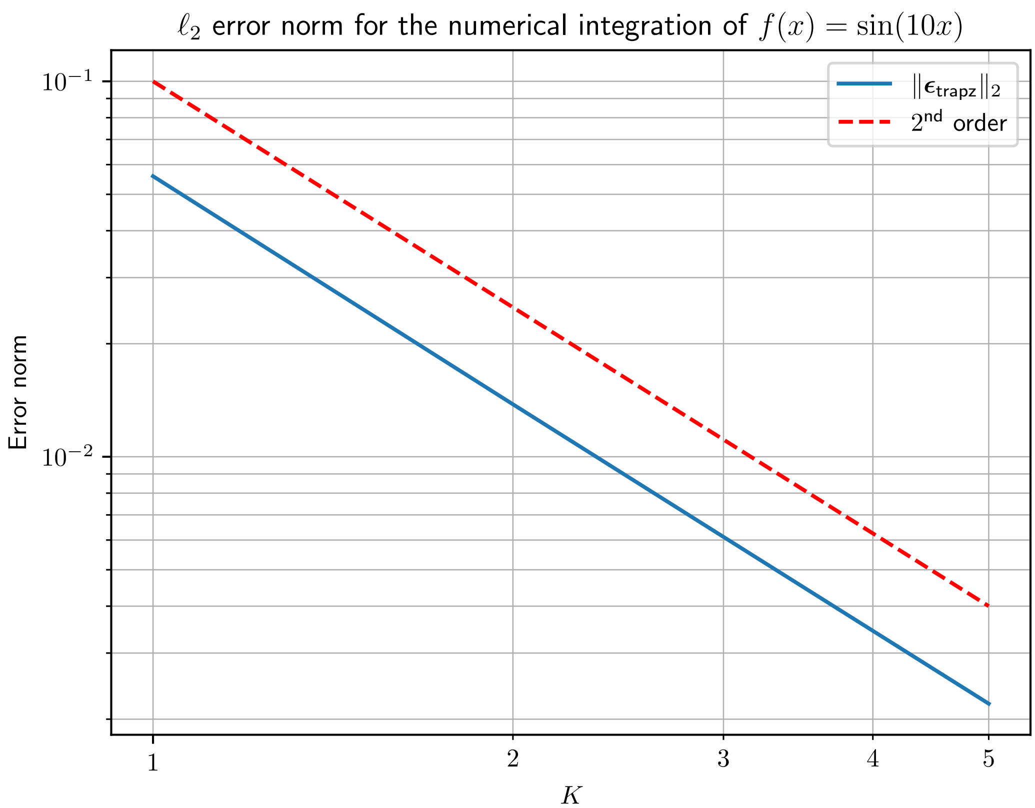
on the computational domain $[0, 1]$ discretized into $N_x = 10$ cells (i.e., the first cell spans $[0, 1/N_x]$, the second cell spans $[1/N_x, 2/N_x]$, etc.).

Try using $K = 1$, $K = 2$, $K = 3$, or $K = 5$ integration points per cell. Quantitatively comment on the accuracy of your numerical integration against the exact analytical cell-averages of the function.

Solution: The exact average of $\mathcal{F}(x) = \sin(10x)$ on an interval $[a, b]$ is given by

$$\frac{1}{b-a}\int_a^b \mathcal{F}(x)\ \mathrm{d}x = \frac{1}{10(b-a)}(\cos(10a) - \cos(10b))$$

Calculating the ℓ_2 norm of the difference between this exact average and the one obtained with the trapezoidal integration rule, we obtain the following figure:



This shows that the cell averages converge towards the exact solution with 2^{nd} order as K is increased.

Q3.3 → **This question is only required for those taking AE 410/CSE 461 for 4 credit hours.** Develop a computational function that returns the average of any function $\mathcal{F}(x)$ on an interval $[a, b]$ using the Gauss-Legendre integration rule:

$$\frac{1}{b-a}\int_a^b \mathcal{F}(x)\ \mathrm{d}x \simeq \frac{1}{2}\sum_{k=0}^{K-1}\left[w_k\,\mathcal{F}\left(a+\frac{(b-a)}{2}(x_k+1)\right)\right] , \tag{11}$$

where the abscissae and weights $(x_k, w_k), k \in \{0, \dots, K-1\}$, are given by the Gauss-Legendre quadrature rule of order K.

Try using $K = 1$, $K = 2$, $K = 3$, or $K = 5$ quadrature points per cell. Quantitatively comment on the accuracy of your numerical integration against the exact analytical cell-averages of the function.

Hints: In `Python`, you can compute the abscissae and weights corresponding to the Gauss-Legendre quadrature rule of order K using the `Numpy` function:

```
x, w = np.polynomial.legendre.leggauss(K)
```

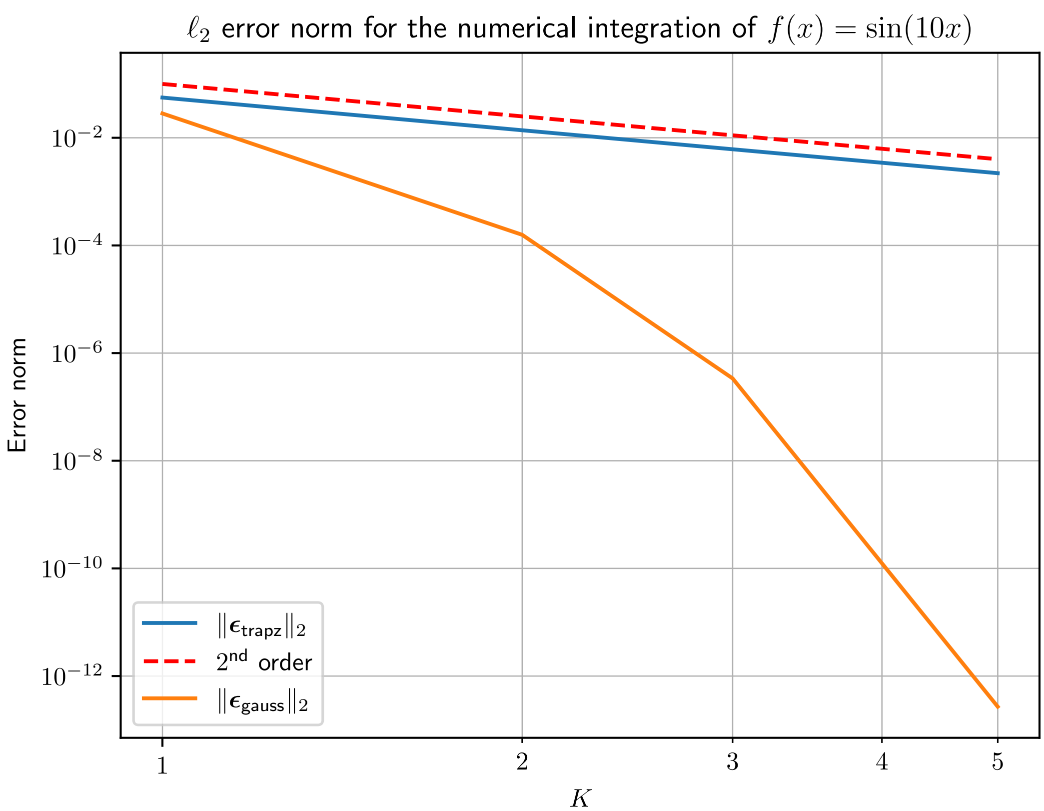
Alternatively, exact Gauss-Legendre abscissae and weights can be found at:

`https://mathworld.wolfram.com/Legendre-GaussQuadrature.html`.

Solution: The exact average of $\mathcal{F}(x) = \sin(10x)$ on an interval $[a, b]$ is given by

$$\frac{1}{b-a}\int_a^b \mathcal{F}(x)\ \mathrm{d}x = \frac{1}{10(b-a)}(\cos(10a) - \cos(10b))$$

Calculating the ℓ_2 norm of the difference between this exact average and the one obtained with the trapezoidal and Gauss-Legendre integration rules, we obtain the following figure:



This shows that the cell averages computed with the Gauss-Legendre integration rule converge with a super-linear rate.

Problem 4 · Write your own finite-volume solver for Burgers' equation using the space- and time-integrated form of the governing equation in each computational cell,

$$U_n^{m+1} - U_n^m + \frac{\Delta t}{\Delta x}\left(F_{n+1/2} - F_{n-1/2}\right) = 0 \ , \tag{12}$$

where all variables have the meanings discussed in class and given in `AE410-Notes-6.pdf`. According to Godunov's method, compute the time averaged fluxes from the solution of localized Riemann problems at the cell faces.

Boundary conditions can be imposed via the introduction of "ghost" cells. At an outflow boundary, the ghost cell value can be extrapolated from the interior of the domain. For instance, at a right outflow boundary, you can use

$$U_{N_x} = U_{N_x-1} \quad \text{or} \quad U_{N_x} = 2U_{N_x-1} - U_{N_x-2} \ . \tag{13}$$

At an inflow boundary, since we do not provide Dirichlet boundary conditions, the ghost cell value can be chosen so as to impose the gradient of u at the boundary to be zero. For instance, at a left inflow boundary, you can use

$$U_{-1} = U_0 \ . \tag{14}$$

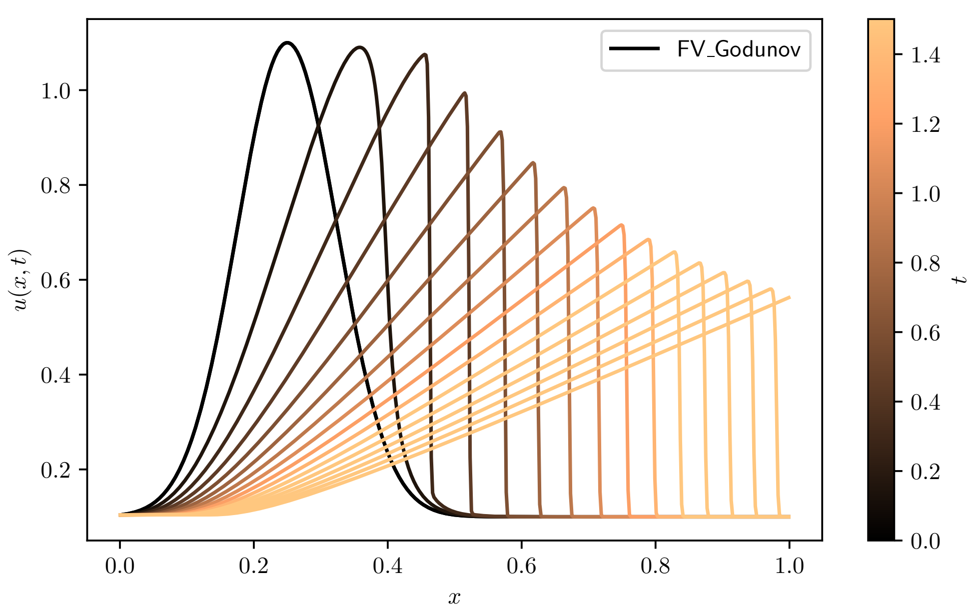
Q4.1 → Initialize your simulation by computing the cell averages of

$$u(x,0) = \frac{1}{10} + \exp\left(\frac{-\left(x-\frac{1}{4}\right)^2}{2\sigma^2}\right) \ , \quad \sigma = \frac{3}{40} \ , \tag{15}$$

on the domain $[0,1]$, discretized into $N_x = 500$ cells.

Q4.2 → With this initial condition, plot the solution to Burgers' equation at times $t \in \{0, 0.1, 0.2, \ldots, 1.5\}$. Make sure that the CFL number is always smaller than 1/2.

Solution: In this problem, we assemble the previously built tools to build a finite volume solver for Burgers' equation. With the inital condition defined in Q4.1, we obtain the following solution:



Q4.3 → **This question is optional. If successfully answered, it will grant you bonus points.**
On the same figure as in Q4.2, plot the analytical solution for those times, until a shock forms.

Solution: In order to verify our solver, we make use of knowing the analytical solution as shown in Problem 1 (keeping in mind that this is only valid in the limit of no shocks being present),

$$u(x,t) = u_0(x - tu(x,t)) \ . \tag{16}$$

Replacing u_0 by its expression given in Q4.1, it follows that

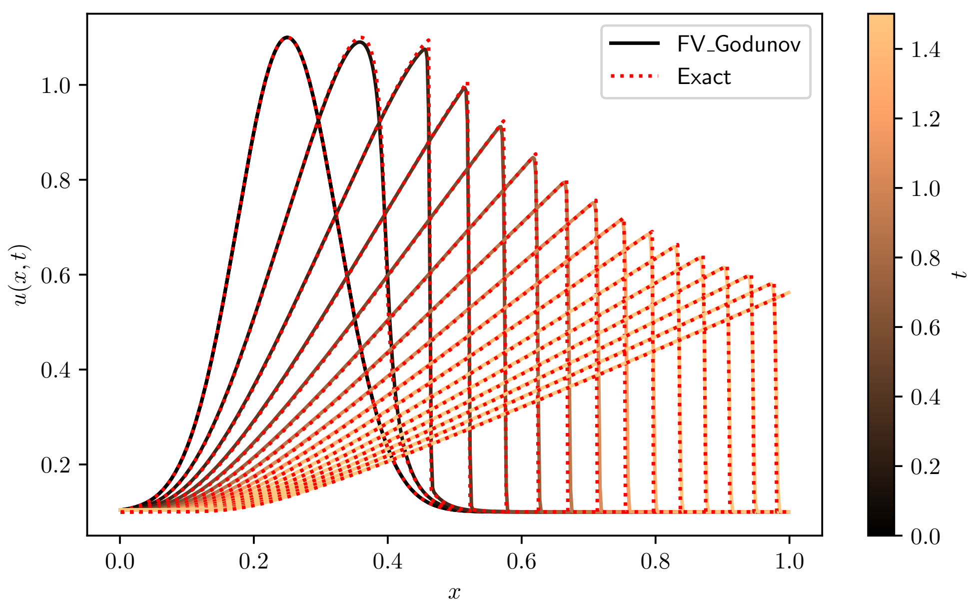
$$u(x,t) = \frac{1}{10} + \exp\left(\frac{-\left(x - tu(x,t) - \frac{1}{4}\right)^2}{2\sigma^2}\right) \ . \tag{17}$$

For any given x and t, $u(x,t)$ can therefore be found as the root of the function

$$u(x,t) - \frac{1}{10} - \exp\left(\frac{-\left(x - tu(x,t) - \frac{1}{4}\right)^2}{2\sigma^2}\right) = 0 \ . \tag{18}$$

In `Python`, such a root can be found using the function `fsolve` of the `SciPy` package.

Adding this analytical to the previous plot, we obtain the following figure:



Hint: You can make use of your result from Problem 1.

Solution: Full codes:

Python (0-based indexing)

```
#!/usr/bin/env python
# coding: utf-8

# Problem 2.2: Riemann solver for Burgers' equation
```

```python

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
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}')

# Riemann solver as derived in class
def RiemannSolver(uL, uR, xi):
  if (uL > uR):
    S = (uL+uR)/2
    if xi < S:
      return uL
    else:
      return uR
  else:
    if xi < uL:
      return uL
    elif xi > uR:
      return uR
    else:
      return xi

# Let's test our solver for the 5 fundamental solutions to this Riemann problem
N = 500
xi = np.linspace(-5, 5, N, endpoint=True)
u = np.zeros((5,N))
labels = ['Right-moving shock', 'Left-moving shock', 'Right-moving fan', 'Left-moving fan'
    , 'Centered fan']
uL = [2, -1, 2/3, -5/3, -1/2]
uR = [1, -2, 5/3, -2/3, 1/2]

# Test the various combinations:
for j in range(len(uL)):
  for i in range(N): u[j][i] = RiemannSolver(uL[j], uR[j], xi[i])

# Plot u(xi) for each case
fig, ax = plt.subplots(1, 1, figsize=(6, 4))
for i in range(5): ax.plot(xi,u[i],label=labels[i])
plt.xlabel(r'$\xi$');plt.ylabel(r'$u(\xi)$');
plt.legend(loc='upper left')
plt.savefig('RiemannSolver_Test1.pdf')
plt.show()

# Plot u(xi) in the x-t plane for each case
fig, ax = plt.subplots(1, 5, figsize=(10, 2), sharey=True)
x = np.linspace(-5, 5, N, endpoint=True); t = np.linspace(1e-9, 5, N, endpoint=True); X, T
    = np.meshgrid(x, t)
u_xt = np.zeros((5,N,N))
for i in range(5):
  ax[i].contourf(X,T,np.interp(X/T, xi, u[i]))
  ax[i].title.set_text(labels[i])
  ax[i].set_xlabel(r'$x$')
  if i == 0: ax[i].set_ylabel(r'$t$')
plt.savefig('RiemannSolver_Test2.pdf')
plt.show()


# Problem 3: Computing cell averages

# Test function to be integrated
def f_test(x):
  return np.sin(10*x)

```

```python
# Exact average of test function on [a,b]
def f_test_avg_exact(a,b):
  return (np.cos(10*a)-np.cos(10*b))/(10*(b-a))

# Numerical average of a function between [a,b] with the trapezoidal rule
def f_avg_num_trapz(f,a,b,K):
  f_avg = 0
  for k in range(K): f_avg += f(a+(b-a)*(k+1)/K) + f(a+(b-a)*k/K)
  f_avg /= 2*K
  return f_avg

# Numerical average of a function between [a,b] with the Gauss-Legendre rule
def f_avg_num_gauss(f,a,b,K):
  xgauss, wgauss = np.polynomial.legendre.leggauss(K)
  f_avg = 0
  for k in range(K): f_avg += wgauss[k]*f(a+(b-a)*(xgauss[k]+1)/2)
  f_avg /= 2
  return f_avg

# Let's test our numerical integration:
Nx = 10
x = np.linspace(0, 1, Nx+1, endpoint=True); dx = x[1] - x[0]
Ks = [1,2,3,5]
avg_exact = np.zeros(Nx)
avg_trapz = np.zeros((4,Nx))
avg_gauss = np.zeros((4,Nx))
for i in range(Nx):
  a = x[i]; b = x[i+1]
  avg_exact[i] = f_test_avg_exact(a,b)
  for j in range(len(Ks)):
    K = Ks[j]
    avg_trapz[j][i] = f_avg_num_trapz(f_test,a,b,K)
    avg_gauss[j][i] = f_avg_num_gauss(f_test,a,b,K)

# Compute L2 and Linf norm of integration error
error_trapz = np.zeros((4,Nx)); L2_trapz = np.zeros(4)
error_gauss = np.zeros((4,Nx)); L2_gauss = np.zeros(4)
for i in range(len(Ks)):
  error_trapz[i] = avg_trapz[i] - avg_exact
  error_gauss[i] = avg_gauss[i] - avg_exact
  L2_trapz[i] = np.sqrt(np.sum(error_trapz[i]**2)/Nx)
  L2_gauss[i] = np.sqrt(np.sum(error_gauss[i]**2)/Nx)

# Plot averaging error norms as a function of K
plt.title(r'$\ell_2$ error norm for the numerical integration of $f(x) = \sin(10x)$')
plt.xlabel(r'$K$');plt.ylabel(r'Error norm')
plt.loglog(Ks,L2_trapz,label=r'$\|\boldsymbol{\epsilon}_\text{trapz}\|_2$')
plt.loglog([1,5],[1e-1,1e-1/25],'r--',label=r'$2^\text{nd}$ order')
# plt.loglog(Ks,L2_gauss,label=r'$\|\boldsymbol{\epsilon}_\text{gauss}\|_2$')
plt.grid(True, which="both", ls="-", linewidth=0.5)
plt.legend()
ax = plt.gca()
ax.xaxis.set_major_formatter(mpl.ticker.ScalarFormatter())
ax.xaxis.set_major_formatter(mpl.ticker.FormatStrFormatter(r'$%d$'))
ax.xaxis.set_minor_formatter(mpl.ticker.ScalarFormatter())
ax.xaxis.set_minor_formatter(mpl.ticker.FormatStrFormatter(r'$%d$'))
plt.savefig('integration_trapz.pdf')
plt.show()


# Problem 4: Finite volume solution of Burgers' equation

# Define flux function
def Flux(u): return u**2/2

# Finite-volumes fluxes for Burgers using Godunov's method
```

```python
def Burgers1D_FV_Godunov(U,dx):
  Nx = len(U)
  # Return F(t,U)
  F = np.zeros(Nx+1)
  for i in range(1,Nx):
    F[i] = Flux(RiemannSolver(U[i-1],U[i],0))
  # Left boundary
  if (U[0] < 0):
    F[0] = Flux(RiemannSolver(2*U[0]-U[1],U[0],0))
  else:
    F[0] = Flux(RiemannSolver(U[0],U[0],0))
  # Right boundary
  if (U[Nx-1] > 0):
    F[Nx] = Flux(RiemannSolver(U[Nx-1],2*U[Nx-1]-U[Nx-2],0))
  else:
    F[Nx] = Flux(RiemannSolver(U[Nx-1],U[Nx-1],0))
  return (F[:Nx] - F[1:])/dx

# Definte initial condition function
def InitialCondition(x):
  return 1/10 + np.exp(-(x-0.25)**2/(2*(3/40)**2))

############## Choose CFL = |u_max|*dt/dx
CFL = 0.5
#########################################

# Initialize case parameters
L = 1; T = 1.5;Nx = 500
x = np.linspace(0, L, Nx+1, endpoint=True); dx = x[1] - x[0]
xc = np.linspace(0.5*dx, L-0.5*dx, Nx, endpoint=True)
U0 = np.zeros(Nx)
for i in range(Nx):
  a = x[i]; b = x[i+1]
  U0[i] = f_avg_num_gauss(InitialCondition,a,b,K)

# Define timestep based on CFL
dt = CFL * dx / np.max(np.abs(U0))

# Explicit Euler time integration
t = 0
U_FV = [U0]
while t <= T:
  # Check CFL
  CFL = dt * np.max(np.abs(U0)) / dx
  if (CFL > 1/2): print("Warning: CFL > 1/2")
  # Euler-FV-Godunov
  U_FV.append(np.copy(U_FV[-1]))
  U_FV[-1] = U_FV[-2] + dt * Burgers1D_FV_Godunov(U_FV[-2],dx)
  t += dt

# Import root-funding algorithm to compute the exact solution
from scipy.optimize import fsolve

# Function to be solved equal to 0 so as to get the exact solution
def ExactSolutionProblem4(u,x,t):
  return u - 1/10 - np.exp(-(x-t*u-0.25)**2/(2*(3/40)**2))

# Plot solutions
fig, ax = plt.subplots(1, 1, figsize=(7, 4))
ax.plot(xc,U0,color=plt.cm.copper(0))
print_legend = True
Nt = len(U_FV)
t = 0; tprint = 0
print_exact = True
for m in range(Nt):
  if t - 0.5*dt <= tprint and tprint < t + 0.5*dt:
```

```
    if print_exact:
      U_Exact = np.zeros(Nx)
      for i in range(Nx):
        U_Exact[i] = fsolve(ExactSolutionProblem4, U_FV[m][i], args=(xc[i],t), xtol=1e-6)
            [0]
    if print_legend:
      ax.plot(xc,U_FV[m],color=plt.cm.copper(m*dt),label='FV_Godunov')
      if print_exact: ax.plot(xc,U_Exact,':r',label='Exact')
      print_legend = False
    else:
      ax.plot(xc,U_FV[m],color=plt.cm.copper(m*dt))
      if print_exact: ax.plot(xc,U_Exact,':r')
    tprint += 0.1
  t += dt
plt.xlabel(r'$x$');plt.ylabel(r'$u(x,t)$');
plt.legend()
fig.colorbar(mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, Nt*dt), cmap='copper'),ax=
    ax, orientation='vertical', label=r'$t$')
plt.savefig('fv_solution_and_exact.pdf')
plt.show()
```

Matlab (1-based indexing)

```
clear all; close all; clc

N = 1000;
eta = linspace(-30,30,N);

% left-moving shock
uL  = -1;
uR  = -2;
for i=1:N
    u1(i) = Riemann_solver( uL,uR, eta(i) );
end

% right-moving shock
uL  = 2;
uR  = 1;
for i=1:N
    u2(i) = Riemann_solver( uL,uR, eta(i) );
end

% left-moving fan
uL  = -1.8;
uR  = -0.8;
for i=1:N
    u3(i) = Riemann_solver( uL,uR, eta(i) );
end

% right-moving fan
uL  = 0.8;
uR  = 1.8;
for i=1:N
    u4(i) = Riemann_solver( uL,uR, eta(i) );
end

% centered fan
uL  = -0.6;
uR  =  0.6;
for i=1:N
    u5(i) = Riemann_solver( uL,uR, eta(i) );
end

figure(1), clf; hold on
```

```
plot(eta,u2,'-','LineWidth',2,'Color',[71, 75, 201]/255)
plot(eta,u1,'-','LineWidth',2,'Color',[242, 153, 19]/255)
plot(eta,u4,'-','LineWidth',2,'Color',[23, 128, 47]/255)
plot(eta,u3,'-','LineWidth',2,'Color',[163, 39, 39]/255)
plot(eta,u5,'-','LineWidth',2,'Color',[153, 84, 199]/255)

xlim([-5,5])
ylim([-2,2.5])
set(gca,'FontSize',20)

xlabel('$\eta$','FontSize',34,'Interpreter','Latex')
ylabel('$u$','FontSize',34,'Interpreter','Latex')

leg=legend('right-moving shock','left-moving shock','right-moving fan',...
           'left-moving fan','centered fan');
set(leg,'FontSize',20,'Location','NorthWest')
set(gcf,'Position',[100 300 800 500])
set(gcf,'PaperPositionMode','Auto')
print(gcf,'P2','-djpeg90')

%%
labelfont = 20;
tickfont  = 12;

tmp = eta;
figure(2), clf;

for i=1:N;
    x(:,i) = tmp(i)*ones(N,1);
    t(i,:) = tmp(i)*ones(N,1);
    u1s(i,:) = u1;
    u2s(i,:) = u2;
    u3s(i,:) = u3;
    u4s(i,:) = u4;
    u5s(i,:) = u5;
end
etas = x./t;
for i=1:N
    soln1(i,:) = interp1(eta,u1,etas(i,:));
    soln2(i,:) = interp1(eta,u2,etas(i,:));
    soln3(i,:) = interp1(eta,u3,etas(i,:));
    soln4(i,:) = interp1(eta,u4,etas(i,:));
    soln5(i,:) = interp1(eta,u5,etas(i,:));
end

a(1)=subplot(1,5,2); hold on
contourf(x,t,soln1);
contour(x,t,soln1);
set(gca,'FontSize',tickfont)

a(2)=subplot(1,5,1); hold on
contourf(x,t,soln2);
contour(x,t,soln2);
set(gca,'FontSize',tickfont)
ylabel('t','FontSize',labelfont)

a(4)=subplot(1,5,4); hold on
contourf(x,t,soln3);
contour(x,t,soln3);
set(gca,'FontSize',tickfont)

a(5)=subplot(1,5,3); hold on
contourf(x,t,soln4);
contour(x,t,soln4);
set(gca,'FontSize',tickfont)

```

```
a(6)=subplot(1,5,5); hold on
contourf(x,t,soln5);
contour(x,t,soln5);
set(gca,'FontSize',tickfont)

linkaxes(a,'xy')
xlim([-10,10])
ylim([0,10])
set(gcf,'Position',[100 300 1200 200])
set(gcf,'PaperPositionMode','Auto')
%print(gcf,'HW4_P2_2','-djpeg90')
```

```
close all

% Let's test our numerical integration:
Nx = 50;
x = linspace(0, 1, Nx+1); dx = x(2) - x(1);
x = x-dx
Ks = [1,2,3,5];
avg_exact = zeros(Nx,1);
avg_trapz = zeros(4,Nx);
avg_gauss = zeros(4,Nx);
for i = 1:Nx
    a = x(i); b = x(i+1);
    avg_exact(i) = f_test_avg_exact(a,b);
    for j = 1:length(Ks)
        K = Ks(j);
        avg_trapz(j,i) = f_avg_num_trapz(@f_test,a,b,K);
        avg_gauss(j,i) = f_avg_num_gauss(@f_test,a,b,K);
    end
end

% Compute L2 and Linf norm of integration error
error_trapz = zeros(4,Nx); L2_trapz = zeros(4,1);
error_gauss = zeros(4,Nx); L2_gauss = zeros(4,1);
for i = 1:length(Ks)
    error_trapz(i,:) = avg_trapz(i,:) - avg_exact';
    error_gauss(i,:) = avg_gauss(i,:) - avg_exact';
    L2_trapz(i) = sqrt(sum(error_trapz(i,:).^2)/Nx);
    L2_gauss(i) = sqrt(sum(error_gauss(i,:).^2)/Nx);
end

% Plot averaging error norms as a function of K
figure;
title('$\ell_2$ error norm for the numerical integration of $f(x) = \sin (10x)$','
    Interpreter','Latex');
loglog(Ks,L2_trapz);
hold on;
loglog([1,5],[1e-1,1e-1/25],'r--');
grid on; grid minor;
xlabel('$K$','Interpreter','latex'); ylabel('Error norm');
leg=legend('$\vert \mathbf{\epsilon}_{trapz} \vert_2$','$2^{nd}$ order');
set(leg,'Interpreter','latex','FontSize',20)
ax = gca;
ax.XTick = Ks;
ax.XTickLabel = cellstr(num2str(Ks'));
saveas(gcf,'P3_1.png');

% Plot averaging error norms as a function of K
figure;
title('$\ell_2$ error norm for the numerical integration of $f(x) = \sin (10x)$','
    Interpreter','Latex');
loglog(Ks,L2_trapz);hold on;
loglog(Ks,L2_gauss);
loglog([1,5],[1e-1,1e-1/25],'r--');
grid on; grid minor;
```

```
xlabel('$K$','Interpreter','latex'); ylabel('Error norm');
leg=legend('$\vert \mathbf{\epsilon}_{trapz} \vert_2$','$2^{nd}$ order');
set(leg,'Interpreter','latex','FontSize',20)
ax = gca;
ax.XTick = Ks;
ax.XTickLabel = cellstr(num2str(Ks'));
saveas(gcf,'P3_2.png');

% Test function to be integrated
function y = f_test(x)
    y = sin(10*x);
end

% Exact average of test function on [a,b]
function y = f_test_avg_exact(a,b)
    y = (cos(10*a)-cos(10*b))/(10*(b-a));
end

% Numerical average of a function between [a,b] with the trapezoidal rule
function f_avg = f_avg_num_trapz(f,a,b,K)
    f_avg = 0;
    for k = 1:K
        %f_avg = f_avg + f(a+(b-a)*(k+1)/K) + f(a+(b-a)*(k-0)/K);
        f_avg = f_avg + f(a+(b-a)*(k+0)/K) + f(a+(b-a)*(k-1)/K);
    end
    f_avg = f_avg / (2*K);
end

% Numerical average of a function between [a,b] with the Gauss-Legendre rule
function f_avg = f_avg_num_gauss(f,a,b,K)
    [xgauss, wgauss] = lgwt(K,a,b);
    f_avg = sum(f(xgauss).*wgauss);
end

% Gauss-Legendre quadrature weights and nodes
function [x,w]=lgwt(N,a,b)
    x = zeros(N,1);
    w = zeros(N,1);
    eps = 1e-15;
    for i = 1:N
        x0 = cos(pi*(i-0.25)/(N+0.5));
        while true
            P1 = 1;
            P2 = 0;
            for j = 1:N
                P3 = P2;
                P2 = P1;
                P1 = ((2*j-1)*x0*P2-(j-1)*P3)/j;
            end
            dP = N*(x0*P1-P2)/(x0^2-1);
            dx = P1/dP;
            x0 = x0 - dx;
            if abs(dx) < eps
                break;
            end
        end
        x(i) = 0.5*(b-a)*x0 + 0.5*(b+a);
        w(i) = 0.5*(b-a)/((1-x0^2)*dP^2);
    end
    w = w*100;
end
```

```
clear all; close all; clc

%% input parameters
Nx  = 500;
```

```
t0   = 0;
xa   = 0;
xb   = 1;

%% derived parameters
L    = xb-xa;
T    = 1.5;
Nt   = 5000; %200;
t    = linspace(t0,T,Nt);
dt   = t(2)-t(1);

%% spatial grid
dx   = L/Nx;
x_grid   = linspace(xa,xb,Nx+1);
x_center= linspace(xa+dx/2,xb-dx/2,Nx);
x_moving= x_center;

%% initial condition
u_grid   = 0.1+exp(-(x_grid-0.25).^2/(2*(3/40)^2));
u        = cellaverage( x_grid,u_grid );

%% store initial condition in solution matrix
us(:,1)      = u;
u_an(:,1)    = u;

%% loop over time
for i=1:Nt
    % check time step
    CFL = max(u)*dt/dx;
    if(CFL>0.5)
        disp(['CFL=',num2str(CFL,'%0.2f'),'!'])
    end

    % fluxes
    for ix=2:Nx
        uL      = u(ix-1);
        uR      = u(ix);
        F(ix)   = Godunov(uL,uR);
    end

    % left boundary
    if(u(1)<0)
        uL = 2*u(1)-u(2);
    else
        uL = u(1);
    end
    uR      = u(1);
    F(1)    = Godunov(uL,uR);

    % right boundary
    if(u(Nx)>0)
        uR = 2*u(Nx-1)-u(Nx-2);
    else
        uR = u(Nx-1);
    end
    uL      = u(Nx);
    F(Nx+1) = Godunov(uL,uR);

    % integrate in time
    for ix=1:Nx
        u(ix)   = u(ix)-dt/dx*(F(ix+1)-F(ix));
    end

    % store unsteady solution in matrix
    us(:,i+1) = u;
end
```

```matlab

%% compute analytical unsteady solution
for i=1:Nt
    if(mod(i,4)==0)
        disp(['Solving nonlinear analytical solution for t=',num2str(t(i),'%0.2f')])
    end
    tt = t(i);
    for ix=1:Nx
        x = x_center(ix);
        if(i==1)
            tmp = cellaverage( x_grid,u_grid );
            u0 = tmp(ix);
        else
            u0 = u_ans(ix,i-1);
        end
        fun = @(u) 0.1+exp(-(x-u*tt-0.25)^2/(2*(3/40)^2) )-u;
        u_ans(ix,i) = fzero(fun,us(ix,i));
    end
end

%% plot unsteady solution and save in video file
while(1)
fname = ['P4_solution'];
v = VideoWriter(fname,'MPEG-4');
v.FrameRate = 30;
open(v);
figure(1), clf;
for i=1:10:Nt
    plot(x_center,us(:,i),'LineWidth',2); hold on
    plot(x_center,u_ans(:,i),'r--','LineWidth',2)
    ylim([0,1.5])
    title(['t=',num2str(t(i),'%0.2f')])
    set(gca,'FontSize',18)
    xlabel('x','FontSize',30,'Interpreter','Latex')
    leg=legend('FV solver','analytical');
    set(leg,'FontSize',24,'Interpreter','Latex','Location','NorthWest')
    frame = getframe(gcf);
    writeVideo(v,frame);
    pause(0.001)
    hold off
end
close(v);
break
end

%% plot unsteady solution in steady plot
figure(2), clf; hold on
no_plots = 20;
for i=1:no_plots
    a(1)=plot(x_center,us(:,(i-1)*Nt/no_plots+1),'LineWidth',2,'Color',[i/no_plots,0,0]);
        hold on
    if(t((i-1)*Nt/no_plots+1)<1.5)
        a(2)=plot(x_center,u_ans(:,(i-1)*Nt/no_plots+1),'b--','LineWidth',2);
    end
end

set(gca,'FontSize',18)
xlabel('x','FontSize',30,'Interpreter','Latex')
leg=legend(a,'FV solver','analytical');
set(leg,'FontSize',24,'Interpreter','Latex','Location','NorthWest')

fname = ['P4_soln'];
set(gcf,'PaperPositionMode','Auto')
set(gcf,'Position',[100 400 800 400])
print(gcf,fname,'-djpeg90')
```

```
function [ u ] = Riemann_solver( uL,uR, eta )
if(uL>uR)
    % shock
    S = 0.5*(uL+uR);
    if(eta>S)
        u = uR;
    else
        u = uL;
    end
else
    % expansion
    if(eta<uL)
        u = uL;
    elseif(eta>uR)
        u = uR;
    else
        u = eta;
    end
end
end
```

```
function [ fc ] = cellaverage( x,f )
    dx=x(2)-x(1);
    for i=1:length(f)-1
        fc(i)    = 1/dx*trapz([x(i),x(i+1)],[f(i),f(i+1)]);
    end
end
```

```
function [ F ] = flux( u )
    F=0.5*u^2;
end
```

```
function [ F ] = Godunov( uL,uR )
    u_shock      = (uL+uR)/2;
    if(uL >= uR)
        if(u_shock>0)
            F = flux(uL);
        else
            F = flux(uR);
        end
    else
        if(uL>0)
            F = flux(uL);
        elseif(uR<0)
            F = flux(uR);
        else
            F = 0;
        end
    end
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