1 Isothermal 1D hydrodynamics solver: sound pulse *(7 points)*

1. 1. Use the 1D finite-volume solver that you already developed for the last problem set to solve the following 1D isothermal hydrodynamics problem: the x-grid goes from $x = -1$ to $x = 1$ and it is divided into $N_x = 100$ cells, the boundary conditions are periodic and the isothermal sound speed is $c_s = 1$.

The initial conditions are

$$\rho(x, t = 0) = 1 + \varepsilon \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{1}$$

$$u(x, t = 0) = 0 \tag{2}$$

where $\varepsilon = 10^{-4}$ and $\sigma = 0.2$. Use CFL=0.4.

For the integration, we utilize the pre-written code made available on the Moodle website. First, we need to do the parameter initialization.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import argrelextrema
from tqdm import tqdm

from source import Grid, Pars, time_loop


EPSILON = 1e-4
SIGMA = 0.2

# initialize parameters
pars = Pars()
pars.Nx = 100
pars.x1 = -1
pars.x2 = 1
pars.cs = 1
pars.cfl = 0.4

# intitial conditions
x = np.linspace(pars.x1, pars.x2, pars.Nx)
rho_0 = 1 + EPSILON * np.exp(-x**2 / (2*SIGMA**2))
u_0 = np.array([0] * pars.Nx)
```

Next, we'd like to implement a function that allows us to easily get $\rho(x)$ and $u(x)$ for various integration times $t_{\max}$. For this, the code base is expanded by the following:

```
def integrate(tmax):
    pars.tmax = tmax

    # create grid and set initial conditions
    grid = Grid(pars)
    grid.cons = np.array([rho_0, u_0])

    # run
    time_loop(grid, pars)
    rho, u = grid.cons
    return rho, u
```

2. How do you expect the system to physically evolve? Why?

If our expectations are valid, the initially very localized density distribution will disperse, i.e. spread out. For high values of $t_{\max}$, we expect the density to equalize across x. The density $\rho(x)$ approximates a constant value that is independent of location.

Due to the periodic boundary conditions, the distribution does not just spread out into infinity, and $\rho(x)$ does not fall monotonously with time. Instead, the boundary conditions lead to a periodic reappearance of the initial distribution's structure (on time scale t_{c_s}).

3. What is the value of the sound crossing time scale t_{c_s} for this setup?

To find t_{c_s}, we calculate $\rho(x = 0)$ for various values of $t_{\max}$. Plotting them, the sound crossing time scale can be determined by averaging over the distance between two peaks of the periodic structure:

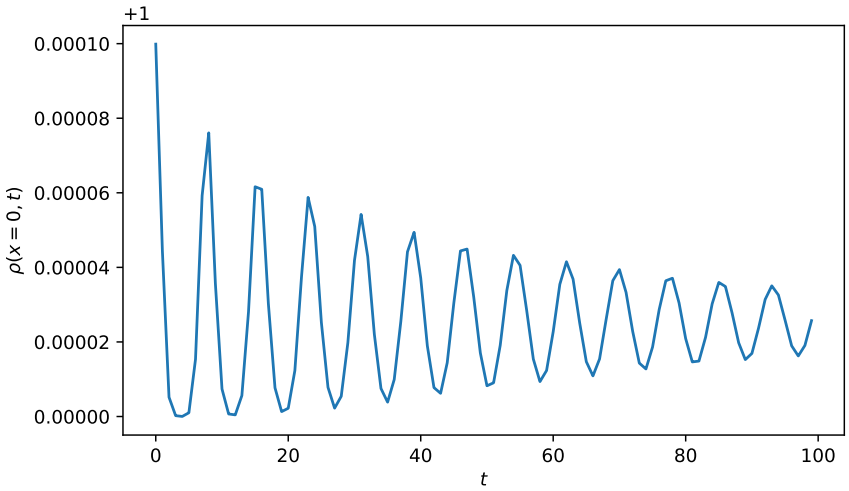
```python
def find_tcs():
    rhos = []
    for tmax in tqdm(np.linspace(0, 25, 100)):
        rho, u = integrate(tmax)
        rhos.append(rho[int(pars.Nx / 2)])  # get rho(x=0)

    plt.figure(figsize=(7, 4))
    plt.plot(rhos)
    plt.xlabel('$t$')
    plt.ylabel(r'$\rho(x=0,t)$')
    plt.savefig('../figures/sound_crossing_time_scale.pdf')
    plt.close()

    # get positions of maxima, the determine t_cs
    max_ind = argrelextrema(np.array(rhos), np.greater)[0]
    tcs = np.mean(np.diff(max_ind))
    return tcs
```

This yields

$$t_{c_s} \approx 7.727$$



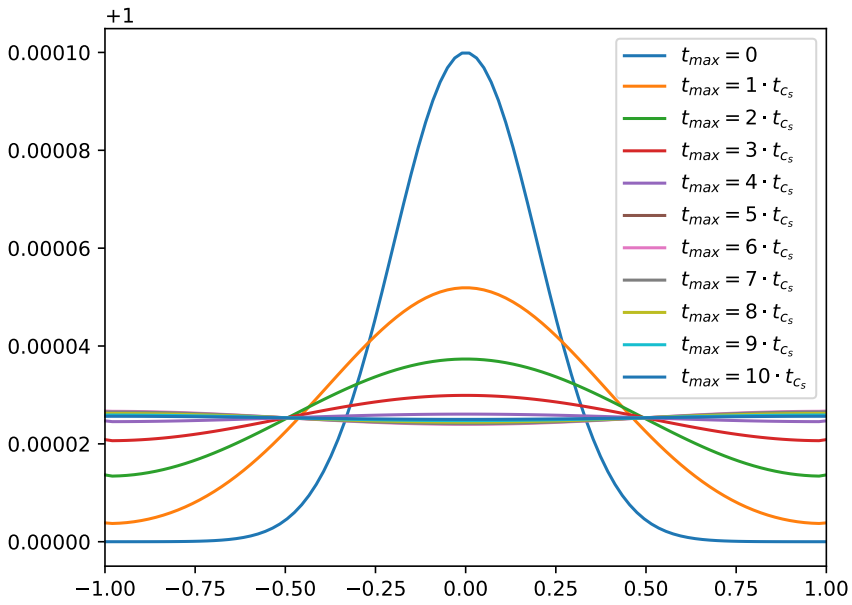
4. Overplot $\rho(x, 0)$ and $\rho(x, t^*)$ for $t^* = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) \times t_{c_s}$. You will need these overplots to make a qualitative comparison, so don't worry if you didn't store the output exactly at t^*, also the previous or the following time step will be fine.

Python code for getting density distribution for different values of $t_{\max}$ and plotting them:

```
for i in tqdm(range(11)):
    tmax = i * tcs
    rho, u = integrate(tmax)
    label = r'$t_{max}=' + str(i) + '\cdot t_{c_s}$'
    label = r'$t_{max}=0$' if not i else label
    plt.xlim(x[0], x[-1])
    plt.plot(x, rho, label=label)

plt.legend()
plt.savefig('../figures/density_evolution.pdf')
plt.close()
```

Resulting plot:



As one can see here, our expectation about the long-term behavior of the density distribution seems to be correct. The bell curve flattens and the peak at $x = 0$ drops off. Due to the periodic dropoff behavior of the system, when looking at the time dependence it is important to use mutliples of the sound crossing time scale, otherwise this graph would look very different.

5. Plot the time evolution of the total kinetic energy until $t_{\max} = 10 \times t_{c_s}$.

The (volume-normalized) kinetic energy can be calculated from

$$E_{\text{kin}} = \frac{1}{2}\rho u^2 \tag{3}$$

We do this for every cell and take the sum:

```
E_kin = []
for i in tqdm(range(11)):
    tmax = i * tcs
    rho, u = integrate(tmax)
    E_kin.append(np.sum(rho * u**2 / 2))

plt.scatter(range(11), E_kin)
plt.xlabel('time $t/t_{c_s}$')
plt.ylabel(r'kinetic energy $E_{kin}}$')
plt.savefig('../figures/energy_evolution.pdf')
plt.close()
```

The result can be seen below: