Dan Card
dcard@umich.edu
Aerospace 523: Computational Fluid Dynamics
Homework: 4
Due: November 13th, 2020

1 Burger's Equation

In this problem you will solve Burgers equation, $u_t + f_x = 0,\ f = \frac{1}{2}u^2,\ x \in [0,4)$, periodic boundaries, with the initial condition shown below.

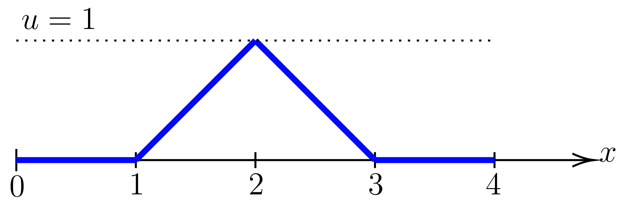


Figure 1: Initial condition to Burgers equation.

For the numerical method, use the finite volume method (FVM) with N_x uniform cells, forward Euler time stepping, a uniform time step, CFL=0.8 (based on the initial condition), and the upwind flux,

$$\hat{F}_{j+\frac{1}{2}} = \frac{1}{2}\left(f_j + f_{j+1}\right) - \frac{1}{2}|\hat{a}_{j+\frac{1}{2}}|\left(u_{j+1} - u_j\right)$$

a. Prior to implementing the FVM, determine the analytical solution using the method of characteristics. Plot the state, $u(x,t)$, at times $t = 0.5,\ 1.0,\ 1.5$ in one figure. In a separate figure, make a space-time diagram of the characteristics, up to $t = 1.5$, and indicate any shock speeds/paths.

Firstly, to start out numbering the regions. Region 1 will be from $0 \to 1$, Region 2 from $1 \to 2$, Region 3 from $2 \to 3$, Region 4 from $3 \to 4$. This gives,

Region 1: $u(x,0) = 0 \quad \forall\ x \in [0,1)$
Region 2: $u(x,0) = x - 1 \quad \forall\ x \in [1,2)$
Region 3: $u(x,0) = 3 - x \quad \forall\ x \in [2,3)$
Region 4: $u(x,0) = 0 \quad \forall\ x \in [3,4)$

This gives that the Initial condition can be expressed as

$$u(x,t=0) = \begin{cases} 0 & x \in [0,1] \\ x-1 & x \in [1,2] \\ 3-x & x \in [2,3] \\ 0 & x \in [3,4] \end{cases}$$

Continued on the next page . . .

Then this gives that the solution to Burgers Equation is,

$$u(x,t) = u_0(x - ut)$$

Where in Region 2, the expression can be given by,

$$\frac{dx}{dt} = x_0 - 1$$

$$x = x_0 + t(x_0 - 1), \; x_0 = \frac{x+t}{1+t}$$

Then again for Region 3, the expression is given by,

$$\frac{dx}{dt} = 3 - x_0$$

$$x = x_0 + t(3 - x_0), \; x_0 = \frac{x-3t}{1-t}$$

Substituting back into the final cases for the characteristics and equations gives,

$$\boxed{u(x,t\le 1) = \begin{cases} 0 & x\in[0,1] \\ \frac{x+t}{1+t} - 1 & x\in[1,2] \\ 3 - \frac{x-3t}{1-t} & x\in[2,3] \\ 0 & x\in[3,4] \end{cases}, \qquad \frac{dx}{dt} = \begin{cases} 0 & x\in[0,1] \\ x_0 + t(x_0-1) & x\in[1,2] \\ x_0 + t(3-x_0) & x\in[2,3] \\ 0 & x\in[3,4] \end{cases}}$$

Then, for the case when a shock forms,

$$s = \frac{dx_s}{dt} = \frac{1}{2}(u_l + \cancelto{0}{u_r}) = \frac{1}{2}\left(\frac{x+t}{1+t} - 1\right)$$

Then performing a simple differential equatoin with the initial condition $x_s(1) = 3$, when the shock forms gives that the expression for the shocks location is,

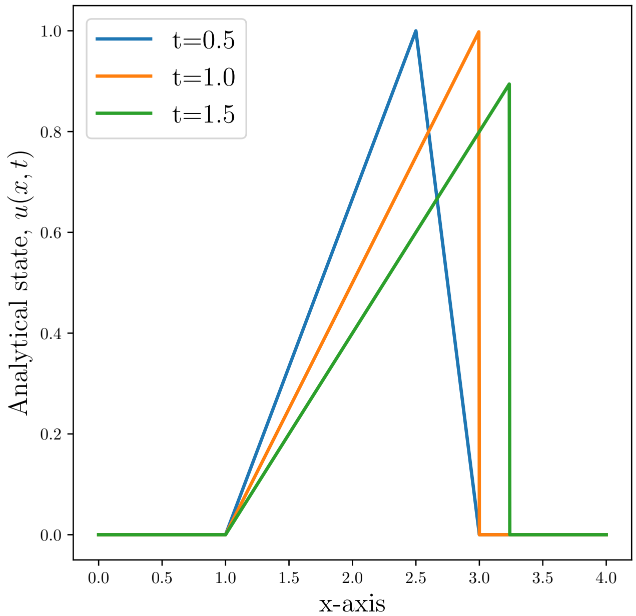
$$x_s = 1 + \sqrt{2+2t}$$

At this moment the right side of the shock will zero or $u_r = 0$, so then then at times higher than the shock forms the expression is then

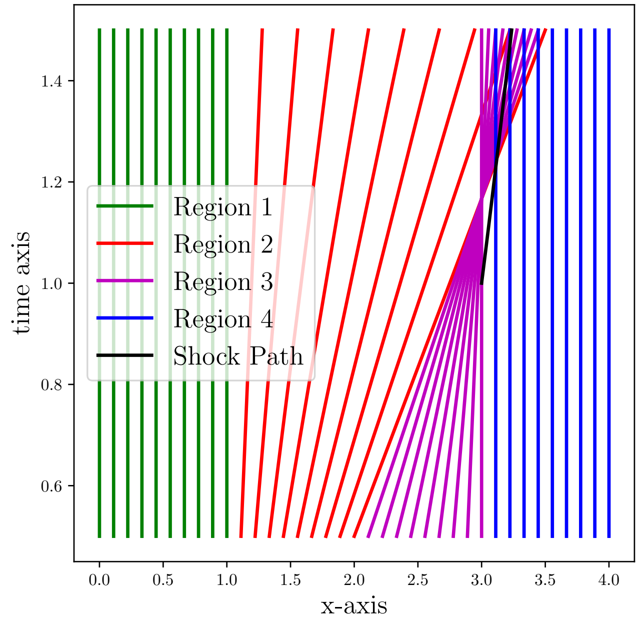
$$\boxed{u(x,t>1) = \begin{cases} 0 & x\in[0,1] \\ \frac{x+t}{1+t} - 1 & x\in[1, 1+\sqrt{2+2t}] \\ 0 & x\in[1+\sqrt{2+2t}, 4] \end{cases}}$$

Continued on the next page . . .

Plotting the characteristics, and the state gives the following results,



(a) Analytical expression for Burgers equation at varying times.

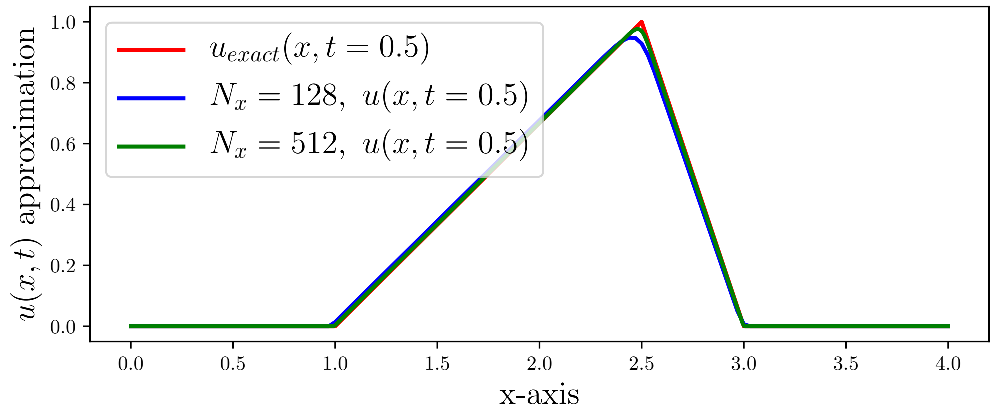


(b) Characteristics of the initial condition of the Burgers equation.

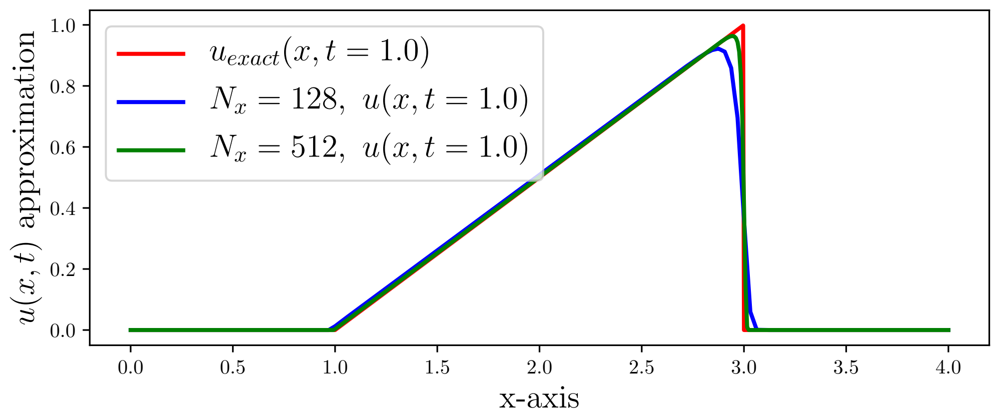
Figure 2: Plots of the analytical Burgers equation and its characteristics at varying times t.

As shown above in Figures 2a, 2b are the implementations of the analytical solution and the characteristics in Python. Shown above in 2a is the actual shock paths as the time varies. At time $t = 1$, the shock first forms the discontinuity. Denoted in Figure 2b by the black line is the shock path following the line $x_s = 1 + \sqrt{2 + 2t}$ for times greater than "1" after the shock forms.

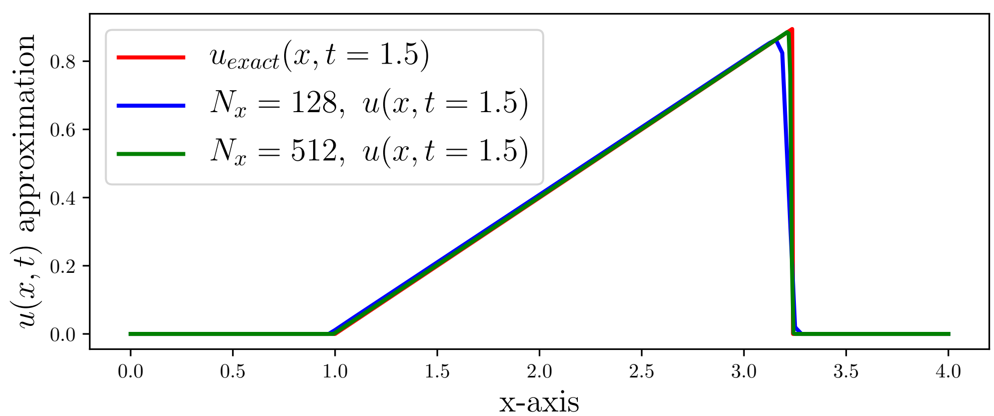
b. Implement the FVM and using N_x= 128 and N_x= 512, show the states at the same times as requested in the previous part. Make three plots, one for each time, and overlay the two N_x results and the analytical solution on each plot. Comment on the differences.



(a) Burgers equation approximated solutions at time $t = 0.5$ for $N_x = 128,\ N_x = 512$.



(b) Burgers equation approximated solutions at time $t = 1.0$ for $N_x = 128,\ N_x = 512$.



(c) Burgers equation approximated solutions at time $t = 1.5$ for $N_x = 128,\ N_x = 512$.

Figure 3: Implementation of the Finite Volume Method (FVM) at varying times.

Shown above in Figure 3 is the approximated solution to Burgers equation and the forming shock. Uniform across all the times, is that at a higher $\mathbf{N_x}$ value there is a better approximation to analytical solution. Shown best in Figure 3b is how the approximated finite volume method bends around the shock arising from the weak solution.

c. Perform a convergence study of the FVM, using the L_2 solution error norm at $t = 0.5$, for $N_x = 128,\ 256,\ 512,\ 1024$. Include an error convergence plot and compute/discuss the rate.

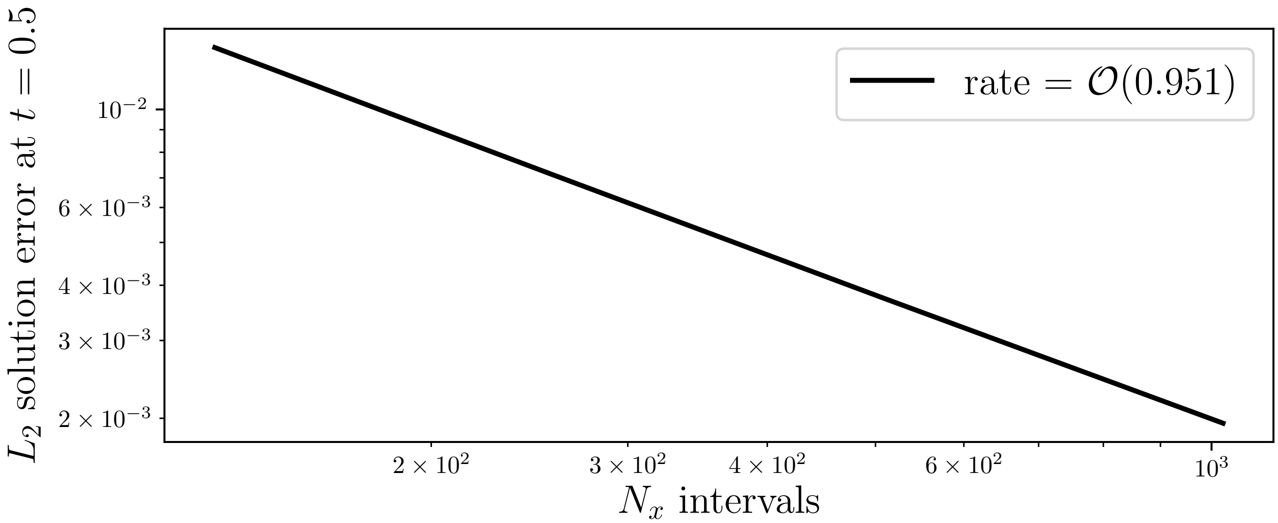


Figure 4: Convergence studies for the finite volume method

Shown above in Figure 4 is the $\mathbf{L_2}$ solution error norm for the finite volume method. As shown in the figure, this implementation has a convergence of $\approx \mathcal{O}(\mathbf{1})$ indicating that it is first-order accurate. Shown below in Table 1 are the converge rates at each interval stepping to the next confirming first-order accuracy.

Table 1: Convergence rates for the finite volume method.

Intervals	Rate
$N_x = 128$	0.951
$N_x = 256$	0.943
$N_x = 512$	0.930

2 Shock Tube

A shock tube consists of two chambers containing air ($\gamma = 1.4$) at different states, as shown below. The domain length is $L = 1$, and the diaphragm is in the middle, at $x = 0.5$.

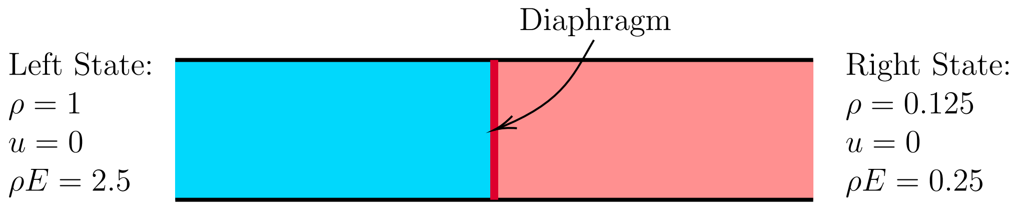


Figure 5: Shock tube chamber configuration.

At $t = 0$, the diaphragm between the two chambers is broken, sending a shock wave and a contact wave into one chamber and an expansion into the other, as shown schematically below for the density.

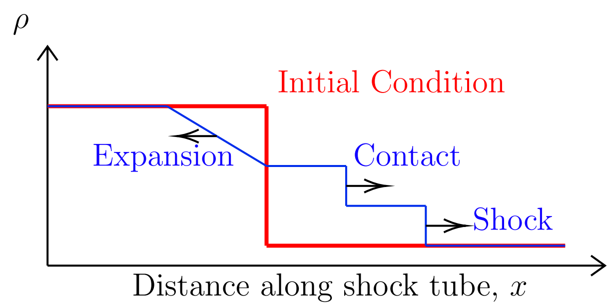


Figure 6: Evolution of unsteady evolution.

In this problem you will use a finite-volume method to simulate the unsteady evolution of the gas state in the shock tube (both chambers). The Euler equations govern the flow, and the units are conveniently chosen to give $\mathcal{O}(1)$ quantities.

a. Write a code that implements a first-order FVM on a uniform grid of N cells, with Dirichlet boundary conditions enforced using the flux function and a constant exterior state, and a constant time step estimated using the CFL condition. Implement both the Rusanov and the HLLE fluxes.Describe your code and ensure that you pass the free-stream preservation test.

b. Run your code to a final time of $T = 0.2$ with both fluxes, on grids of $N = 50,\ 100,\ 200$. Make figures showing the density, momentum, and velocity for each grid, overlaying the two flux function results on each figure (9 figures total). Discuss the behavior of the solution and the differences between the fluxes.

Python Main Driving Code and Analytical Solution

Algorithm 1: Main driving Python code to conduct convergence studies.

```
import matplotlib.pyplot as plt
import numpy as np
import math
from fvm import solve, getglob, flux

plt.rc('text', usetex=True)
plt.rc('font', family='serif')

def q1a():

    plt.figure(figsize=(6,6))
    for t in np.array([0.5, 1.0, 1.5]):
        x0, u0 = analytical(1000, t)
        plot_label = r't=' + str(t)
        plt.plot(x0, u0, lw = 2, label=plot_label)
    plt.xlabel(r'x-axis', fontsize=16)
    plt.ylabel(r'Analytical state, $u(x,t)$', fontsize=16)
    plt.legend(loc='upper left', fontsize=16)
    plt.savefig('analytical.pdf', bbox_inches='tight')
    plt.show()



    plt.figure(figsize=(6,6))
    plt.plot(np.NaN, np.NaN, lw=2, color='g')
    plt.plot(np.NaN, np.NaN, lw=2, color='r')
    plt.plot(np.NaN, np.NaN, lw=2, color='m')
    plt.plot(np.NaN, np.NaN, lw=2, color='b')
    plt.plot(np.NaN, np.NaN, lw=2, color='k')
    characteristics('1')
    characteristics('2')
    characteristics('3')
    characteristics('4')
    t = np.linspace(1, 1.5, num=50, endpoint=True); xs = 1 + np.sqrt(2 + 2*t)
    plt.plot(xs, t, lw=2, color='k')
    plt.xlabel(r'x-axis', fontsize=16)
    plt.ylabel(r'time axis', fontsize=16)
    plt.legend([r'Region 1', r'Region 2', r'Region 3', r'Region 4', r'Shock Path'], loc='center left', fontsize=16)
    plt.savefig('characteristics.pdf', bbox_inches='tight')
    plt.show()

def q1b():

    xex, uex05 = analytical(1000, 0.5)
    xex, uex10 = analytical(1000, 1.0)
    xex, uex15 = analytical(1000, 1.5)



    Nx = 128;
    x128, u0 = getIC(Nx)
    uNx12805 = solve(x128, u0, 0.5, 0.8)
    uNx12810 = solve(x128, u0, 1.0, 0.8)
    uNx12815 = solve(x128, u0, 1.5, 0.8)

    Nx = 512;
    x512, u0 = getIC(Nx)
    uNx51205 = solve(x512, u0, 0.5, 0.8)
    uNx51210 = solve(x512, u0, 1.0, 0.8)
    uNx51215 = solve(x512, u0, 1.5, 0.8)

    plt.figure(figsize=(8,3))
    plt.plot(xex, uex05, lw=2, color='r', label=r'$u_{exact}(x,t=0.5)$')
    plt.plot(x128, uNx12805, lw=2, color='b', label=r'$N_x = 128,\ u(x,t=0.5)$')
    plt.plot(x512, uNx51205, lw=2, color='g', label=r'$N_x = 512,\ u(x,t=0.5)$')
    plt.xlabel(r'x-axis', fontsize=16)
    plt.ylabel(r'$u(x,t)$ approximation', fontsize=16)
    plt.legend(loc='upper left', fontsize=16)
    plt.savefig('t05.pdf', bbox_inches = 'tight')
    plt.show()

```

```
    plt.figure(figsize=(8,3))
    plt.plot(xex, uex10, lw=2, color='r', label=r'$u_{exact}(x,t=1.0)$')
    plt.plot(x128, uNx12810, lw=2, color='b', label=r'$N_x = 128,\ u(x,t=1.0)$')
    plt.plot(x512, uNx51210, lw=2, color='g', label=r'$N_x = 512,\ u(x,t=1.0)$')
    plt.xlabel(r'x-axis', fontsize=16)
    plt.ylabel(r'$u(x,t)$ approximation', fontsize=16)
    plt.legend(loc='upper left', fontsize=16)
    plt.savefig('t10.pdf', bbox_inches = 'tight')
    plt.show()

    plt.figure(figsize=(8,3))
    plt.plot(xex, uex15, lw=2, color='r', label=r'$u_{exact}(x,t=1.5)$')
    plt.plot(x128, uNx12815, lw=2, color='b', label=r'$N_x = 128,\ u(x,t=1.5)$')
    plt.plot(x512, uNx51215, lw=2, color='g', label=r'$N_x = 512,\ u(x,t=1.5)$')
    plt.xlabel(r'x-axis', fontsize=16)
    plt.ylabel(r'$u(x,t)$ approximation', fontsize=16)
    plt.legend(loc='upper left', fontsize=16)
    plt.savefig('t15.pdf', bbox_inches = 'tight')
    plt.show()

def q1c():
    nxs = np.array([128, 256, 512, 1024])
    errs = np.zeros(4); k = 0

    for nx in nxs:
        x, u0 = getIC(nx)
        u = solve(x, u0, 0.5, 0.8)
        xex, uex = analytical(nx, 0.5)

        errs[k] = l2err(u, uex); k += 1

    f = open('convergences', 'w'); output = ''
    for i in range(3):
        rate = abs(math.log10(errs[i+1]/errs[i])/math.log10(nxs[i+1]/nxs[i]))
        output += r'$N_x = $ ' + str.format('{0:.0f}',nxs[i])+ r'& ' + str.format('{0:.3f}', rate)
            + r'\\'
    f.write(output)
    f.close()

    rate = math.log10(errs[1]/errs[0])/math.log10(nxs[1]/nxs[0])
    plotlabel = r'rate = $\mathcal{O}$(' + str.format('{0:.3f}', abs(rate)) + ')'

    plt.figure(figsize=(8,3))
    plt.plot(nxs, errs, lw=2, color='k', label=plotlabel)
    plt.xlabel(r'$N_x$ intervals', fontsize=16)
    plt.ylabel(r'$L_2$ solution error at $t=0.5$', fontsize=16)
    plt.yscale('log')
    plt.xscale('log')
    plt.legend(loc='upper right', fontsize=16)
    plt.savefig('convergence.pdf', bbox_inches = 'tight')
    plt.show()

def l2err(u, uex):
    err = 0
    for i in range(u.shape[0]):
        err += (u[i] - uex[i])**2
    err = np.sqrt(1/u.shape[0] * err)

    return err

def characteristics(region):
    ts = np.array([0.5, 1.0, 1.5]); ints = 10

    if region == '1':
        x = np.linspace(0, 1, num=ints, endpoint=True)
        for i in range(x.shape[0]):
            plt.plot([x[i],x[i]],[ts[0], ts[2]], lw=2, color='g')
    elif region == '2':
        x = np.linspace(1, 2, num=ints, endpoint=True)
        for i in range(1, x.shape[0]):
            plt.plot([x[i],x[i] + ts[2]*(x[i]-1)],[ts[0], ts[2]], lw=2, color='r')
    elif region == '3':
        x = np.linspace(2, 3, num=ints, endpoint=True)
        for i in range(1, x.shape[0]):
            plt.plot([x[i],x[i] + ts[2]*(3 - x[i])],[ts[0], ts[2]], lw=2, color='m')
    else:
        x = np.linspace(3, 4, num=ints, endpoint=True)
```

```python
        for i in range(1, x.shape[0]):
            plt.plot([x[i],x[i]],[ts[0], ts[2]], lw=2, color='b')

def analytical(Nx, t):
    x = np.linspace(0, 4, Nx+1, endpoint=True)
    u = np.zeros(Nx + 1)

    x0 = state_init(x, Nx)
    if t != 1.5:
        for i in range(Nx+1):
            if x[i] >= 0 and x[i] <= 1:
                u[i] = 0
            elif x[i] > 1 and (x[i] + t)/(1 + t) - 1 < (3 - (x[i] - 3*t)/(1 - t)):
                u[i] = (x[i] + t)/(1 + t) - 1
            elif x[i] >= 2 and x[i] < 3:
                if (1 - t) == 0:
                    u[i] == 0
                else:
                    u[i] = 3 - (x[i] - 3*t)/(1 - t)
            elif x[i] >= 3 and x[i] <=4:
                u[i] = 0
    else:
        for i in range(Nx+1):
            if x[i] >= 0 and x[i] <= 1:
                u[i] = 0
            elif x[i] > 1 and x[i] <= 1+np.sqrt(2)*np.sqrt(1+t):
                u[i] = (x[i] + t)/(1 + t) - 1
            else:
                u[i] = 0
    return x, u

def state_init(x, Nx):
    x0 = np.zeros(Nx + 1)
    for i in range(Nx+1):
        if x[i] == 0 or x[i] < 1:
            x0[i] = 0
        elif x[i] == 1 or x[i] < 2:
            x0[i] = x[i] - 1
        elif x[i] == 2 or x[i] < 3:
            x0[i] = 3 - x[i]
        else:
            x0[i] = 0

    return x0

def getIC(Nx):
    x = np.linspace(0, 4, Nx + 1)
    xc = 0.5*(x[0:Nx] + x[1:Nx+1])

    u = np.zeros(Nx+1)
    for i in range(Nx+1):
        if x[i] == 0 or x[i] < 1:
            u[i] = 0
        elif x[i] == 1 or x[i] < 2:
            u[i] = xc[i] - 1
        elif x[i] == 2 or x[i] < 3:
            u[i] = 3 - xc[i]
        else:
            u[i] = 0

    return x, u

if __name__ == "__main__":
    q1a()
    q1b()
    q1c()
```

Python Implementation for Finite Volume Method

Algorithm 2: Implementation of finite volume method and convergence studies.

```python
import numpy as np

def getglob(u):
    a = max(u)
    return a

def flux(ul, ur, a):
    Fhat = 1/2*ul**2
    return Fhat

def solve(x, u0, T, CFL):
    a = getglob(u0)
    dx = x[1] - x[0]
    dt = CFL*dx/a; Nt = int(np.ceil(T/dt)); dt = T/Nt
    Ne = u0.size

    u = u0.copy(); R = u.copy()
    for n in range(Nt):
        R *= 0
        for j in range(Ne+1):
            ul = u[j-1]
            ur = u[j ] if (j < Ne) else u[0]
            Fhat = flux(ul,ur,a)
            if (j > 0 ): R[j-1] += Fhat
            if (j < Ne): R[j ] -= Fhat
        u -= dt/dx * R
    return u
```