Traffic Flow modelling

Avinash Kori —ED15B006
Engineering Design Department,
Indian Institute of Technology, Madras
koriavinash1@gmail.com

November 2, 2018

Contents

1 Introduction

In this report various experiments were conducting considering 1D traffic flow. 1D conservation law was considered along with few other assumptions, like no traffic can enter in between. All the traffic entering from one end should leave from other end, The traffic is assumed to be dense in nature, model uses the solution of continuous differential equations which means dense conditions must satisfy. Traffic flow is considered as the solution for Burger's given by:

$$U_t + F(U)_x = 0$$

, given initial condition as

$$U(x, 0) = U_0(x)$$

Further sections proceeds in understanding mechanism in plotting characteristic equations for the above equation, various numerical schemes to compute the solution for above partial differential equation, and application of above equation in modelling traffic flow around traffic light and speed breakers.

2 Characteristics Plot Implementation

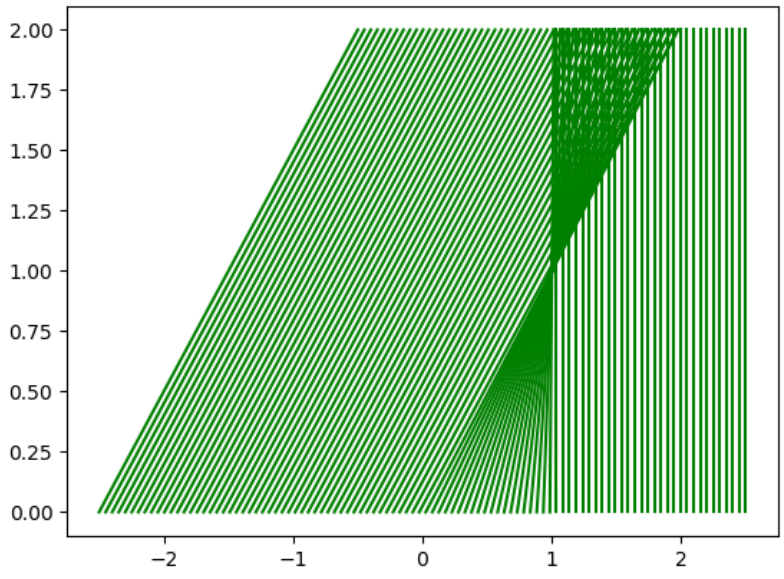
This function shows the implementation of characteristics plotting function for Burger's equation.

```
def characteristic_solution(x0, t):
        x = []
        for i in range(len(x0)):
                x.append(rho2U(initial_conditions(x0[i])*t + x0[i]))
        return np.array(x)


def plot_characteristic(x, t):
        for i in range(x.shape[0]):
                plt.plot(x[i], t, 'g')
        plt.show()
```

Based on given initial conditions characteristics plots are obtained for example:

$$U_0(x_0, t) = \begin{cases} 1 & \text{if } x \leq 0 \\ 1 - x & \text{if } 0 < x \leq 1 \\ 0, & \text{if } x > 1 \end{cases} \tag{1}$$

characteristics curve obtained for the above initial conditions:



3 Numerical Schemes

In this section approximation for continuous differential equation is mentioned using combination of forward and backward difference equations. This section involves the python implementation for all the proposed methods.

3.1 Upwind Method

3.1.1 Methodology

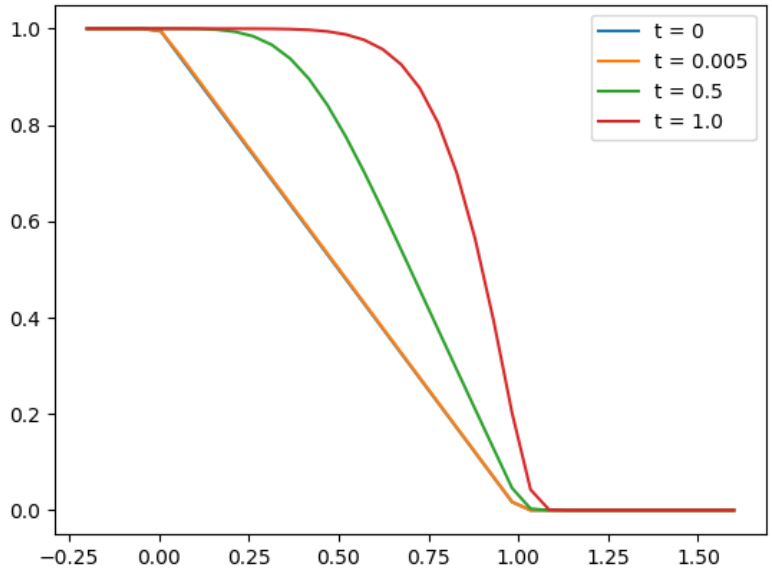
Finite forward and backward difference scheme for upwind method:

```
def Upwind_Method(F, FD, U, k, h):
        temp = np.zeros_like(U)
        for i in range(1, len(U) - 1):
                if FD(U[i]) >= 0:
                        temp[i] = U[i] - (k/h)*(F(U[i]) - F(U[i-1]))
                else:
                        temp[i] = U[i] - (k/h)*(F(U[i +1]) - F(U[i]))
        return temp
```

In the above function F denotes function, which is used in Burger's equation: $F = U^2/2$, FD denoted derivative of F , i.e $FD = U$.

3.1.2 Results

The following plot shows the results U(x,t) for initial condition shown in equation 1, with k = 0.005, and h = 0.14



3.2 Lax-Friedrichs scheme

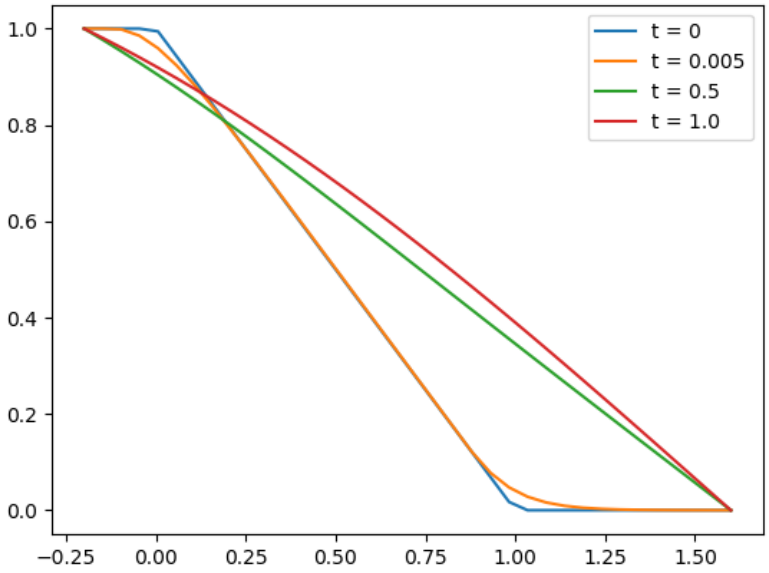
3.2.1 Methodology

```
def Lax_Friedrichs_scheme(F, FD, U, k, h):
        temp = np.zeros_like(U)
        for i in range(1, len(U) - 1):
                temp[i] =0.5*(U[i+1] + U[i-1]) - \
                        (k/(2.*h))*(F(U[i +1]) - F(U[i]))
        return temp
```

In the above function F denotes function, which is used in Burger's equation: $F = U^2/2$, FD denoted derivative of F , i.e $FD = U$.

3.2.2 Results

The following plot shows the results U(x,t) for initial condition shown in equation 1, with k = 0.005, and h = 0.14



3.3 Richtmyer two-step Lax-Wendroff scheme

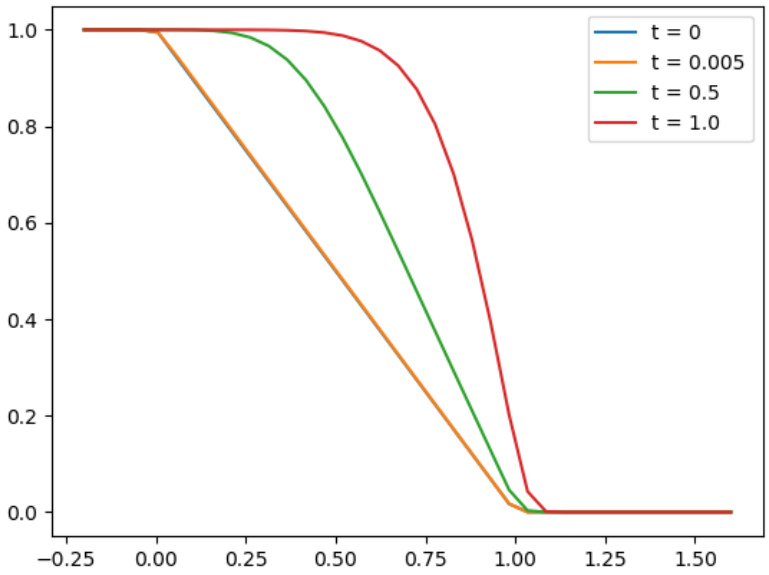
3.3.1 Methodology

```
def Richtmyer_two_step_Lax_Wendroff_scheme(F, FD, U, k, h):
        Ustar = lambda u, up1: 0.5*(u + up1) - (k/h)*(F(up1) - F(u))
        temp = np.zeros_like(U)
        for i in range(1, len(U) - 1):
                temp[i] = U[i] - (k/h)*(F(Ustar(U[i], U[i+1])) - \
                F(Ustar(U[i-1], U[i])))
        return temp
```

In the above function F denotes function, which is used in Burger's equation: $F = U^2/2$, FD denoted derivative of F , i.e $FD = U$.

3.3.2 Results

The following plot shows the results U(x,t) for initial condition shown in equation 1, with k = 0.005, and h = 0.14



3.4 Mac-Cornack scheme

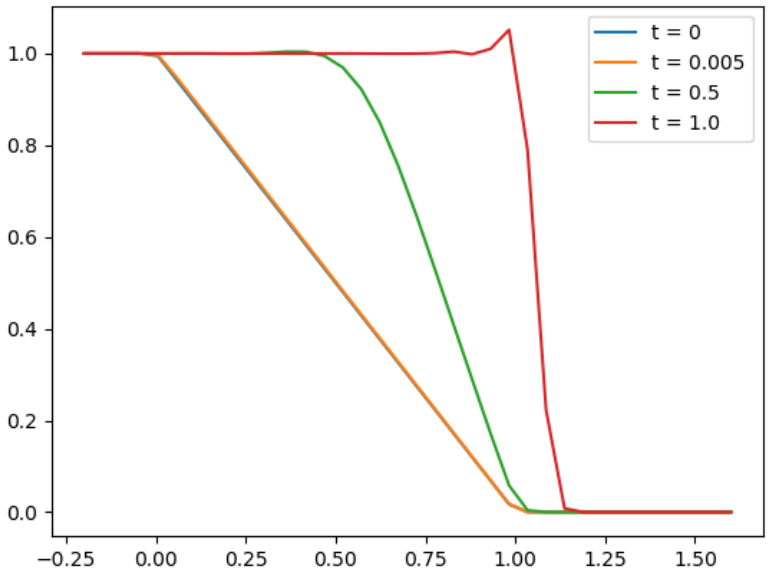
3.4.1 Methodology

```
def Mac_Cornack_scheme(F, FD, U, k, h):
        Ustar = lambda u, up1: u - (k/h)*(F(up1) - F(u))
        temp = np.zeros_like(U)
        for i in range(1, len(U) - 1):
                temp[i] =0.5*(U[i] + Ustar(U[i], U[i+1])) -\
                 (k/h)*(F(Ustar(U[i], U[i+1])) - F(Ustar(U[i-1], U[i])))
        return temp
```

In the above function F denotes function, which is used in Burger's equation: $F = U^2/2$, FD denoted derivative of F , i.e $FD = U$.

3.4.2 Results

The following plot shows the results U(x,t) for initial condition shown in equation 1, with k = 0.005, and h = 0.14



3.5 Gudonov Method

3.5.1 Methodology

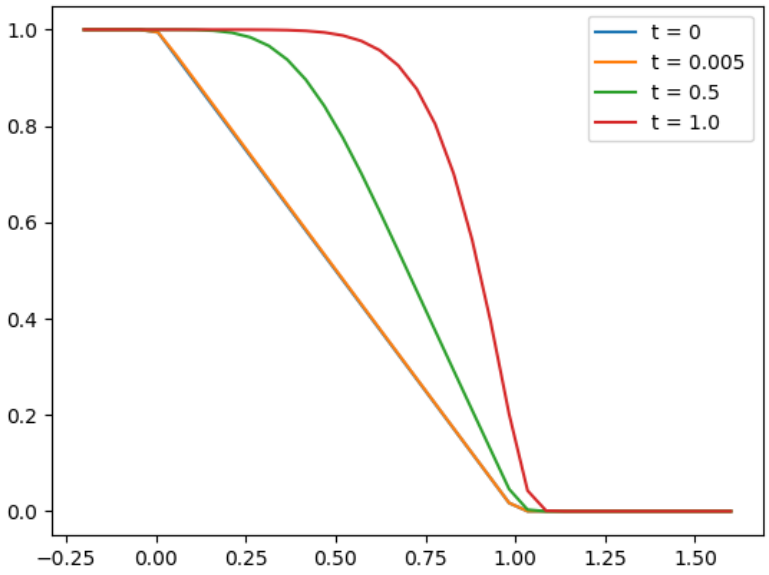
```
def Gudonov_Method(F, FD, FStarSolve, U, k, h):
        def Speed(u, up1):
                return float(F(up1) - F(u))/(up1 - u)

        def Ustar(u, up1):
                if (FD(u) >= 0) and (FD(up1) >= 0):
                        return u
                elif (FD(u) < 0) and (FD(up1) < 0):
                        return up1
                elif (FD(u) >= 0) and (FD(up1) < 0):
                        if Speed(u, up1) >= 0:
                                return u
                        else:
                                return up1
                elif (FD(up1) >= 0) and (FD(u) < 0):
                        return FStarSolve()

        temp = np.zeros_like(U)
```

```
    for i in range(1, len(U) - 1):
            temp[i] = U[i] -(k/h)*(F(Ustar(U[i], U[i+1])) -\
                    F(Ustar(U[i-1], U[i])))
    return temp
```

In the above function F denotes function, which is used in Burger's equation: $F = U^2/2$, FD denoted derivative of F , i.e $FD = U$.

3.5.2 Results

The following plot shows the results U(x,t) for initial condition shown in equation 1, with k = 0.005, and h = 0.14

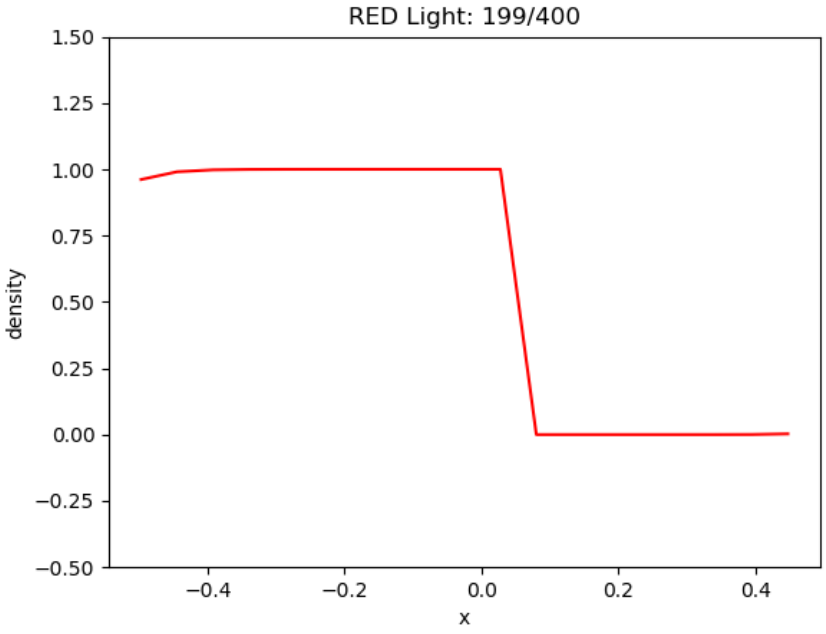


4 Results on Traffic Lights

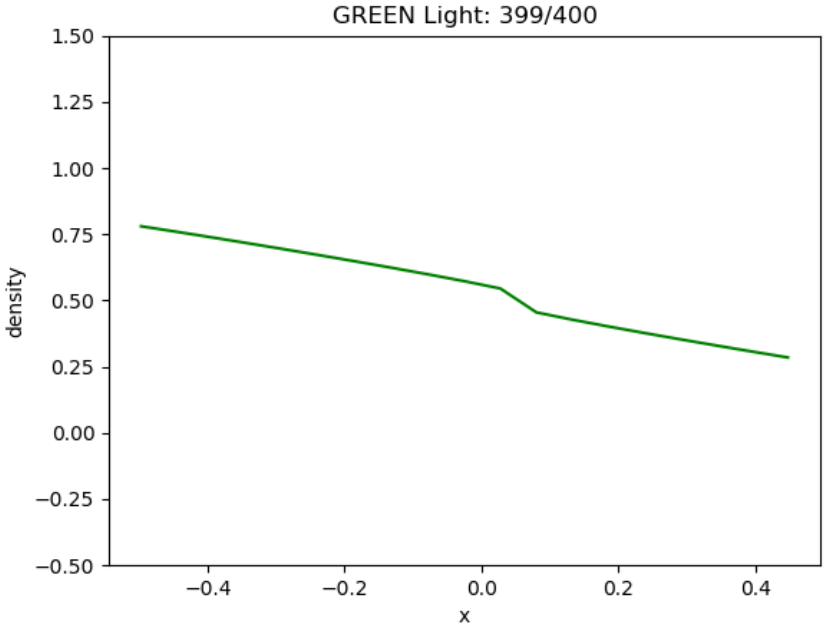
In case of traffic light:

$$U(x,t) = 1 - \frac{2\rho(x,t)}{\rho_{max}}$$

the bellow figure shows the density of traffic before and after red light, and the change in density when light changes from red to green.


RED Light: 199/400
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
density
−0.4
−0.2
0.0
0.2
0.4
x

(a)


GREEN Light: 399/400
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
density
−0.4
−0.2
0.0
0.2
0.4
x

(b)

The animation of change in traffic density is present in imgs folder in github.

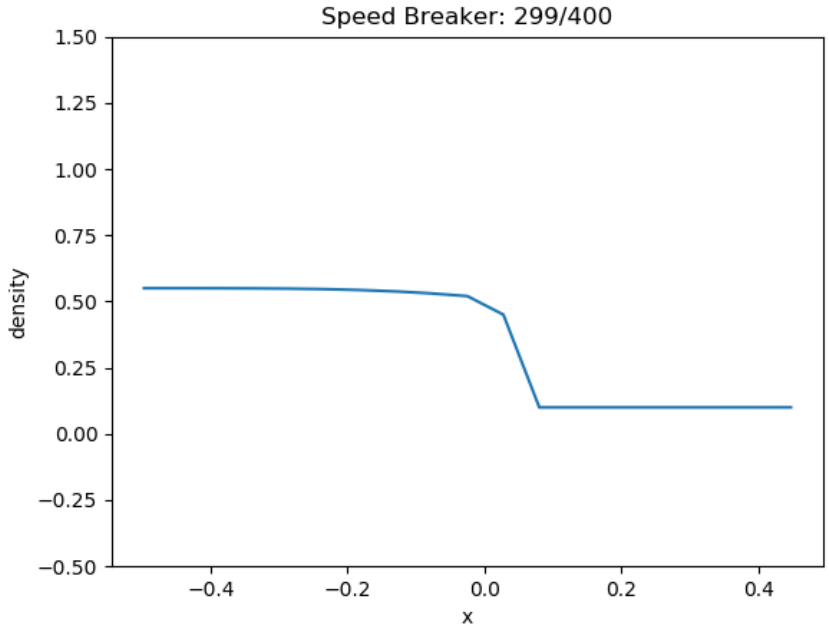
The entire experiment was simulated for about 4sec, with first 2sec of red light and next 2sec of green light. As sub-figure (a) in the above figure denotes the saturation state due to red light i.e density before traffic light is maximum while density after traffic light is 0. Similarly when green light is simulated entire traffic gets distributed throughout to (before and after 0, **0 denotes the location of traffic light**)

5 Results on Speed Breakers

In case of speed breaker:

$$F(x,t) = \rho v_{max}(1 - \frac{\rho(x,t)}{\rho_{max}})$$

$$v(x,t) = v_{max}(1 - \frac{2\rho(x,t)}{\rho_{max}})$$



(c)

The above figure shows the effect of traffic flow when speed breakers are present after signal and signal is changed from red to green. The animation of change in traffic density is present in imgs folder in github.

This experiment was also simulated for 4sec, with velocity before speed breaker being 0.1 units/sec while the velocity after speed breaker being 0.9 units/sec. It can be seen that after time model gets saturated with high traffic density before speed breaker and low density after speed breaker.

6 Code availability and structure

This Report comes with a dedicated GitHub repository where all codes, animations and pre-trained models will be uploaded.
(https://github.com/koriavinash1/MathematicalModelling/TrafficFlow)

Folder Structure of Code:

- TrafficFlow
  - imgs
  - references
  - src
    - traficlight.py
    - test.py
    - speedbreaker.py

7 Conclusions

In this experiments various numerical schemes to solve partial differential equations were evaluated. Godonov being the best scheme for dynamic shock wave prorogation and Riemann problems, was used in analyzing traffic flow with aforementioned initial conditions and hyper-parameters.

8 Acknowledgements

All the codes are implemented using opensource libraries in python:
+ numpy
+ matplotlib
+ Only academic references (class report and provided thesis) were used in formulating all the experiments in this report.