Solving Burger's Equation By RK4

Deepak Singh

June 2024

1 Introduction

The Burgers' equation is a fundamental partial differential equation from fluid mechanics. It combines both nonlinear convection and diffusion, making it an essential model for various physical phenomena. The equation can be written as:

$$u_t + uu_x = \nu u_{xx},$$

where u is the velocity field, t is time, x is the spatial coordinate, and ν is the viscosity coefficient.

To solve this equation numerically using the Runge-Kutta method of fourth order (RK4), we must first discretize the equation in both space and time. The RK4 method is a popular technique for solving ordinary differential equations (ODEs), and we can apply it to the time integration part of the discretized PDE.

2 Steps for Solving Burgers' Equation Using RK4

2.1 Spatial Discretization

- Divide the spatial domain into a grid with points $x_i = i\Delta x$ where $i = 0, 1, 2, \ldots, N$ and Δx is the spatial step size.
- Use finite difference approximations for the spatial derivatives. For example:
  - The first derivative u_x can be approximated using central differences:

    $$u_x \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}.$$

  - The second derivative u_{xx} can be approximated using central differences:

    $$u_{xx} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}.$$

2.2 Time Discretization

- Let the time domain be divided into discrete steps $t_n = n\Delta t$ where $n = 0, 1, 2, \ldots$ and Δt is the time step size.
- Use the RK4 method to update the solution in time.

3 RK4 Scheme for Burgers' Equation

Given the semi-discrete form of Burgers' equation (discretized in space but not yet in time):

$$\frac{du_i}{dt} = -u_i \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \nu \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2},$$

we can define a function $F(u, t)$ that represents the right-hand side of the equation:

$$F(u_i, t) = -u_i \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \nu \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}.$$

Now, apply the RK4 method to this ODE:

3.1 Compute the slopes:

$$\begin{aligned}
k_1 &= \Delta t \, F(u^n, t_n), \\
k_2 &= \Delta t \, F(u^n + \frac{1}{2}k_1, t_n + \frac{1}{2}\Delta t), \\
k_3 &= \Delta t \, F(u^n + \frac{1}{2}k_2, t_n + \frac{1}{2}\Delta t), \\
k_4 &= \Delta t \, F(u^n + k_3, t_n + \Delta t).
\end{aligned}$$

3.2 Update the solution:

$$u^{n+1} = u^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

4 Implementation Outline

1. Initialize the parameters and initial conditions.
2. Loop over time steps:
   - Compute the spatial derivatives using finite difference methods.
   - Calculate the slopes k_1, k_2, k_3, k_4.
   - Update the solution using the RK4 formula.
3. Apply boundary conditions as necessary.

5 Summary

Solving the Burgers' equation using the RK4 method involves discretizing the spatial derivatives using finite difference methods and then integrating the resulting system of ODEs using the RK4 scheme. This method is advantageous due to its higher accuracy and stability compared to lower-order methods, making it suitable for capturing the complex dynamics of Burgers' equation.

6 Python Code

The following Python code demonstrates the numerical solution of the Burgers' equation using the RK4 method:

Listing 1: Python Code for Solving Burgers' Equation

```
import numpy as np

n = 10  # Number of grid points
y0 = np.cos(np.linspace(0, 2*np.pi, n))
t0 = 0  # initial time
t_end = 2  # final time
nu = 0.1  # Viscosity coefficient

# Define the function f(y, t)
def f(y, t, nu=0.1):
    n = len(y)
    dydt = np.zeros_like(y)

    # Compute dy/dt for each point using finite differences
    for i in range(n):
        if i == 0:
            dydt[i] = -y[i] * (y[i+1] - y[n-1]) / 2 + nu * (y[n-1] - 2 * y[i] +
        elif i == n-1:
            dydt[i] = -y[i] * (y[0] - y[i-1]) / 2 + nu * (y[i-1] - 2 * y[i] + y[
        else:
            dydt[i] = -y[i] * (y[i+1] - y[i-1]) / 2 + nu * (y[i-1] - 2 * y[i] +

    return dydt

# Run RK4 method
def RK4(f, n, y0, t0, t_end):
    t_values = [t0]
    y_values = [y0]
    t = t0
    y = y0
    h = (t_end - t0) / n  # step size
    while t < t_end:
        k1 = f(y, t)
        k2 = f(y + 0.5 * h * k1, t + 0.5 * h)
        k3 = f(y + 0.5 * h * k2, t + 0.5 * h)
        k4 = f(y + h * k3, t + h)
        y = y + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
        t = t + h
        t_values.append(t)
        y_values.append(y)
    return np.array(t_values), np.array(y_values)

t_values, y_values = RK4(f, n, y0, t0, t_end)
```

7 Output

The following output was obtained from running the above Python code:

```
At t = 0 value of y is = [ 1.          0.76604444  0.17364818 -0.5        -0.939
 -0.5         0.17364818  0.76604444  1.        ]
for t= 0.2 value of y is = [ 1.01896957  0.83179839  0.19828195 -0.56055162 -0.9767954  -0.8
 -0.44871143  0.15385997  0.70540544  0.97282155]

for t= 0.4 value of y is = [ 1.02813156  0.9019579   0.2294544  -0.63219901 -1.00393829 -0.8
 -0.40532041  0.13777068  0.65039073  0.9395927 ]

for t= 0.6000000000000001 value of y is = [ 1.0263321   0.97492956  0.26952071 -0.71701407 -
 -0.36852488  0.12450979  0.60092067  0.90243711]

for t= 0.8 value of y is = [ 1.01317279  1.04818259  0.32185739 -0.81752954 -1.01882251 -0.7
 -0.33721425  0.11344028  0.55667433  0.86335101]

for t= 1.0 value of y is = [ 0.98913423  1.11804652  0.39137643 -0.93700851 -1.00281008 -0.6
 -0.31044632  0.10409067  0.51719651  0.82405106]

for t= 1.2 value of y is = [ 0.95560192  1.17952005  0.485344   -1.07997456 -0.96959634 -0.6
 -0.28742668  0.09610806  0.48197561  0.78588322]

for t= 1.4 value of y is = [ 0.91478965  1.22610241  0.61473165 -1.25325385 -0.91882513 -0.6
 -0.26748945  0.08922538  0.45049654  0.74978839]

for t= 1.5999999999999999 value of y is = [ 0.86957853  1.24966661  0.79657071 -1.46805224 -
 -0.25007916  0.08323831  0.42227408  0.71631273]

for t= 1.7999999999999998 value of y is = [ 0.8233109   1.24042811  1.05840659 -1.74426719 -
 -0.23473413  0.07798882  0.39687169  0.68564696]
```

```
for t= 1.9999999999999998 value of y is = [ 0.77959231  1.18714043  1.44780212 -2.12016886 -
 -0.22107119  0.07335332  0.37391008  0.65767937]
```

8 Conclusion

This document demonstrates the solution of the Burgers' equation using the RK4 method implemented in Python. The results are presented as values of y at different time steps, showing the evolution of the solution over time.

9 Appendix

Here, we include an image related to the topic for better visualization.

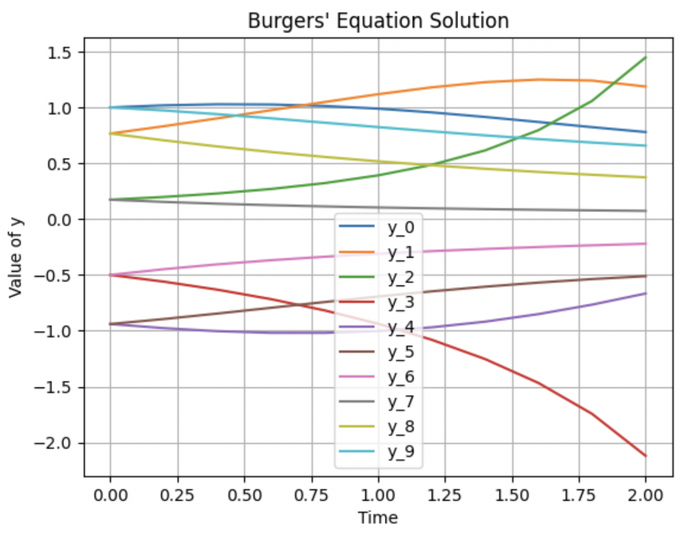

Figure 1: Plotting of solution corresponding to x and y .