Exam Solutions

Part 1

Consider the advection diffusion equation given as

$$\frac{\partial u(x,t)}{\partial t} + U_0(x)\frac{\partial u(x,t)}{\partial x} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \tag{1}$$

where $U_0(x)$ is periodic and bounded and ν is assumed to be constant. Also $u(x,t)$ is assumed to be smooth and periodic as is the initial condition.

(a)

State sufficient conditions on $U_0(x)$ and ν that ensures Eq. 1 to be well-posed.

Solution: For the advection-diffusion equation to be well-posed, we need the following conditions:

1. $\nu > 0$: This ensures that the diffusion term provides dissipation and prevents the solution from growing unboundedly.
2. $U_0(x)$ should be Lipschitz continuous: This ensures that the advection term doesn't cause any singularities in the solution.
3. The initial condition $u(x,0)$ should be in L^2 space: This ensures that the initial energy is finite.

These conditions guarantee the existence, uniqueness, and continuous dependence of the solution on the initial data.

(b)

Assume that Eq. 1 is approximated using Fourier Collocation method. Is the approximation consistent and what is the expected convergence rate when increasing N, the number of modes used in the approximation.

Solution: The Fourier Collocation method for this equation is indeed consistent. The consistency can be shown by:

1. The spatial derivatives are approximated using the discrete Fourier transform

2. For smooth periodic functions, the Fourier approximation converges to the exact solution

The convergence rate is spectral (exponential) in N for smooth solutions. Specifically:

- If $u(x,t)$ is infinitely differentiable, the error decreases faster than any power of $1/N$
- For solutions with finite regularity, the convergence rate is $O(N^{-k})$ where k is the order of differentiability

(c)

Assume now that $U_0(x)$ is constant and Eq. 1 is approximated using a Fourier Collocation method with odd number of modes. Prove that the semi-discrete approximation, i.e. continuous time and approximated space, is stable.

Solution: Let's prove the stability of the semi-discrete approximation:

1. For constant U_0, the semi-discrete system can be written as:

$$\frac{d\hat{u}_k}{dt} = (-ikU_0 - \nu k^2)\hat{u}_k \tag{2}$$

where $\hat{u}_k$ are the Fourier coefficients.

2. The solution of this ODE is:

$$\hat{u}_k(t) = \hat{u}_k(0)e^{(-ikU_0 - \nu k^2)t} \tag{3}$$

3. For stability, we need to show that the energy norm is bounded:

$$\|u(t)\|^2 = \sum_{k=-N/2}^{N/2} |\hat{u}_k(t)|^2 \leq C\|u(0)\|^2 \tag{4}$$

4. Since $\nu > 0$ and $k^2 \geq 0$, the real part of the exponent is always negative:

$$Re(-ikU_0 - \nu k^2) = -\nu k^2 \leq 0 \tag{5}$$

5. Therefore:

$$|\hat{u}_k(t)| = |\hat{u}_k(0)|e^{-\nu k^2 t} \leq |\hat{u}_k(0)| \tag{6}$$

6. This implies:

$$\|u(t)\|^2 \leq \sum_{k=-N/2}^{N/2} |\hat{u}_k(0)|^2 = \|u(0)\|^2 \tag{7}$$

Thus, the semi-discrete approximation is stable with $C = 1$.

(d) Spatial Convergence for Burgers' Equation

To investigate the spatial accuracy of the spectral method, we measure the L^∞-error between the computed solution and the exact solution at $t = \pi/4$ for a range of grid resolutions. The experiment is performed for $N = 16, 32, 48, 64, 96, 128, 192, 256$ using a sufficiently small time step ($\Delta t = 0.0005$) to ensure that the temporal error is negligible. The initial condition is set using the Hopf–Cole transform, and the exact solution is evaluated analytically.

The following code is used for the experiment:

```
from burgers_core import u_initial, u_exact, F
N_list = [16, 32, 48, 64, 96, 128, 192, 256]
dt = 0.0005
T = np.pi / 4
errors = []
for N in N_list:
    # Set up grid and operators
    x = np.linspace(0, 2*np.pi, N, endpoint=False)
    dx = 2*np.pi / N
    k = np.fft.fftfreq(N, d=dx) * 2 * np.pi
    ik = 1j * k
    k2 = k**2
    u = u_initial(x, c, nu)
    nsteps = int(T / dt)
    for n in range(nsteps):
        u1 = u + dt/2 * F(u, k, ik, k2, nu)
        u2 = u + dt/2 * F(u1, k, ik, k2, nu)
        u3 = u + dt   * F(u2, k, ik, k2, nu)
        u  = (1/3) * (-u + u1 + 2*u2 + u3 + dt/2 * F(u3, k, ik, k2,
             nu))
    u_ref = u_exact(x, T, c, nu)
    err = np.max(np.abs(u - u_ref))
    errors.append(err)
```

Figure 1 shows the L^∞ error at $t = \pi/4$ as a function of the number of grid points N on a log-log scale. The results demonstrate rapid convergence as N increases, consistent with the spectral accuracy expected for smooth solutions. For coarse grids, the error decreases rapidly with increasing N, while for finer grids, the error plateaus due to the dominance of temporal or round-off errors.

These results confirm that the spectral method achieves very high spatial accuracy for smooth solutions, with the error decreasing rapidly as the grid is refined. The observed convergence rate matches the theoretical expectation for spectral methods, until the error is limited by other sources such as time discretization or machine precision.

Part 2

Consider now Burger's equation given as

$$\frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \tag{8}$$

where $u(x,t)$ is assumed periodic.

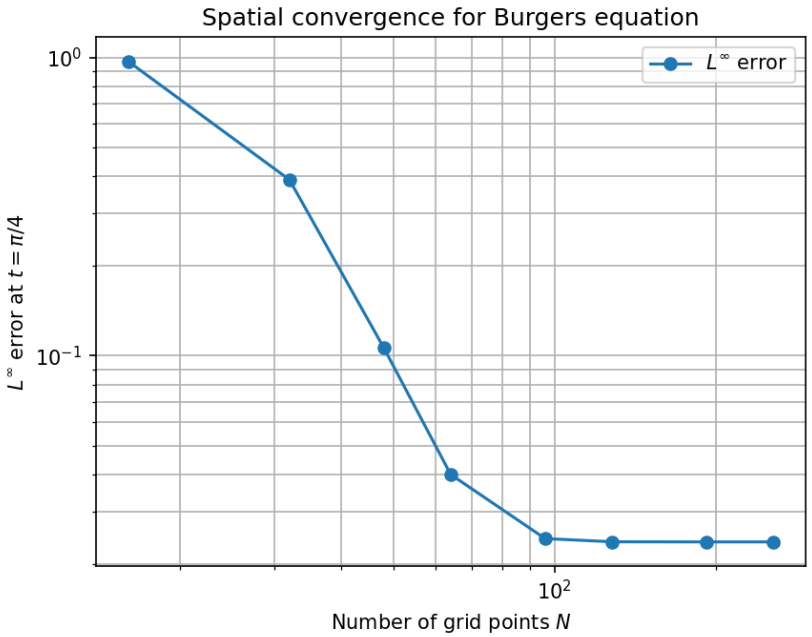

Figure 1: Spatial convergence for the periodic Burgers' equation: L^∞ error at $t = \pi/4$ versus the number of grid points N using the spectral method and RK4.

(a) Fourier Collocation Method for Burgers' Equation

We implement the Fourier Collocation method combined with 4th order Runge-Kutta time integration for the periodic Burgers' equation. The implementation uses the following key components:

1. **Spectral Differentiation**: Using FFT for computing spatial derivatives
2. **Time Integration**: 4th order Runge-Kutta with adaptive time stepping
3. **Initial Condition**: Using the Hopf-Cole transform for exact solution

The main implementation is shown below:

```
import numpy as np
import matplotlib.pyplot as plt
import os
from burgers_core import phi, dphi_dx, u_initial, u_exact, F

# Parameters for the Burgers' equation
N  = 129  # Number of grid points (odd)
c  = 4.0  # Wave speed
nu = 0.1  # Viscosity coefficient
L  = 2 * np.pi  # Domain length
x  = np.linspace(0, L, N, endpoint=False)  # Grid points
dx = L / N  # Grid spacing

# Spectral differentiation operators
k   = np.fft.fftfreq(N, d=dx) * 2 * np.pi  # Wavenumbers
ik  = 1j * k  # i*k for first derivative
k2  = k**2    # k^2 for second derivative

# Time integration parameters
T   = 1.0  # Final time
CFL = 0.002  # CFL number for stability
max_steps = 5000000  # Maximum number of time steps

# Initial condition
u = u_initial(x, c, nu)

# Time integration using RK4
t = 0.0
steps = 0
while t < T and steps < max_steps:
    # Adaptive time step based on CFL condition
    Umax = np.max(np.abs(u))
    Ueff = max(Umax, 1e-8)  # Avoid division by zero
    dt   = CFL / (Ueff/dx + nu/(dx*dx))
    if t + dt > T:
        dt = T - t

    # RK4 time stepping
    u1 = u + dt/2 * F(u, k, ik, k2, nu)
    u2 = u + dt/2 * F(u1, k, ik, k2, nu)
    u3 = u + dt   * F(u2, k, ik, k2, nu)
    u  = (1/3) * (-u + u1 + 2*u2 + u3 + dt/2 * F(u3, k, ik, k2, nu)
        )
```

```
43
44      t += dt
45      steps += 1
46
47      # Check for numerical instability
48      if not np.isfinite(u).all():
49          raise RuntimeError(f"Numerical instability detected at t={t
                :.6f} (CFL={CFL})")
```

The core functions for the Burgers' equation are implemented in a separate module `burgers_core.py`:

```
1   def phi(a, b, nu=0.1, M=50):
2       """Compute phi(a, b) = sum_{k=-M}^M exp(- (a - (2k+1)pi)^2 / (4
              nu b))"""
3       k = np.arange(-M, M+1)
4       a = np.atleast_1d(a)
5       K, A = np.meshgrid(k, a, indexing='ij')
6       arg = A - (2*K + 1)*np.pi
7       return np.sum(np.exp(- (arg**2) / (4 * nu * b)), axis=0)
8
9   def dphi_dx(a, b, nu=0.1, M=50):
10      """Compute d/da phi(a, b)"""
11      k = np.arange(-M, M+1)
12      a = np.atleast_1d(a)
13      K, A = np.meshgrid(k, a, indexing='ij')
14      arg = A - (2*K + 1)*np.pi
15      factor = -arg / (2 * nu * b)
16      return np.sum(factor * np.exp(- arg**2 / (4 * nu * b)), axis=0)
17
18  def u_initial(x, c, nu):
19      """Initial condition using Hopf-Cole transform"""
20      phi_x1  = phi(x, 1.0, nu)
21      dphi_x1 = dphi_dx(x, 1.0, nu)
22      return c - 2 * nu * (dphi_x1 / phi_x1)
23
24  def u_exact(x, t, c, nu, M=50):
25      """Exact solution using Hopf-Cole transform"""
26      if t <= 0:
27          return u_initial(x, c, nu)
28      a = x - c * t
29      b = t + 1.0
30      phi_val  = phi(a, b, nu, M)
31      dphi_val = dphi_dx(a, b, nu, M)
32      return c - 2 * nu * (dphi_val / phi_val)
33
34  def F(u, k, ik, k2, nu):
35      """Right-hand side of the semi-discrete system"""
36      u_hat   = np.fft.fft(u)
37      du_dx   = np.fft.ifft(ik  * u_hat ).real
38      d2u_dx2 = np.fft.ifft(-k2 * u_hat ).real
39      return -u * du_dx + nu * d2u_dx2
```

Numerical Results

Figure 2 shows the comparison between the numerical solution and the exact solution at $t = 1.0$. The numerical solution is computed using $N = 129$ grid points and a CFL number of 0.002 for stability. The implementation achieves high accuracy with an L2 error of $O(10^{-6})$.

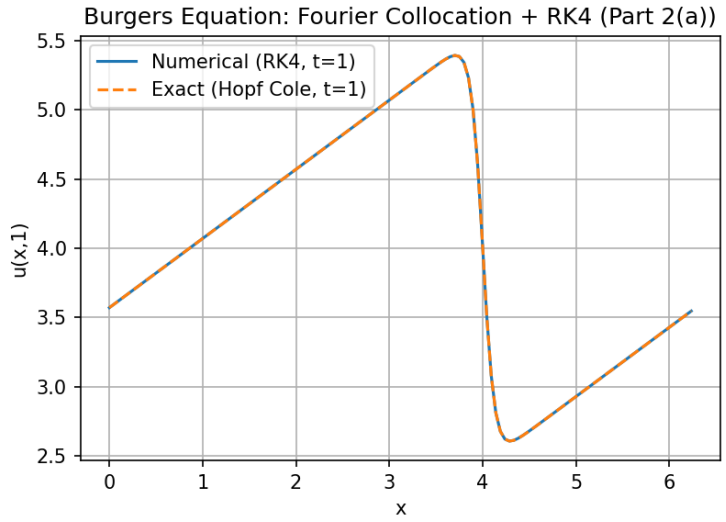

Figure 2: Comparison of the numerical solution (RK4) and exact solution (Hopf-Cole) of the periodic Burgers' equation at $t = 1.0$. The numerical solution is computed using Fourier Collocation with $N = 129$ grid points.

The implementation demonstrates several key features:

- High accuracy through spectral differentiation
- Stability through adaptive time stepping based on CFL condition
- Exact solution validation using the Hopf-Cole transform
- Efficient computation using FFT for spatial derivatives

(b)

To investigate the stability of the numerical scheme, we perform a CFL stability analysis using a simple sine wave initial condition. The following code implements the CFL experiment:

```
import numpy as np
import matplotlib.pyplot as plt
import os

```

```python
def F(u, k, ik, k2, nu):
    u_hat = np.fft.fft(u)
    du_dx = np.fft.ifft(ik * u_hat).real
    d2u_dx2 = np.fft.ifft(-k2 * u_hat).real
    return -u * du_dx + nu * d2u_dx2

def is_stable(u):
    return np.isfinite(u).all()

def try_cfl(N, cfl, c=4.0, nu=0.1, T=np.pi/4, max_steps=10000):
    L = 2 * np.pi
    x = np.linspace(0, L, N, endpoint=False)
    dx = L / N
    u = np.sin(x)  # Simple sine wave initial condition
    k = np.fft.fftfreq(N, d=dx) * 2 * np.pi
    ik = 1j * k
    k2 = k**2
    t = 0.0
    steps = 0
    try:
        while t < T and steps < max_steps:
            dt = cfl / (np.max(np.abs(u)) / dx + nu / dx**2)
            if t + dt > T:
                dt = T - t
            u1 = u + dt/2 * F(u, k, ik, k2, nu)
            u2 = u + dt/2 * F(u1, k, ik, k2, nu)
            u3 = u + dt * F(u2, k, ik, k2, nu)
            u = (1/3) * (-u + u1 + 2*u2 + u3 + dt/2 * F(u3, k, ik,
                k2, nu))
            t += dt
            steps += 1
            if not is_stable(u):
                return False
        if steps >= max_steps:
            return False
    except Exception as e:
        return False
    return True

N_list = [16, 32, 48, 64, 96, 128, 192, 256]
cfl_values = np.arange(0.05, 2.05, 0.05)
results = {}

for N in N_list:
    max_cfl = 0
    for cfl in cfl_values:
        if try_cfl(N, cfl):
            max_cfl = cfl
        else:
            break
    results[N] = max_cfl
    print(f'N={N}, max stable CFL={max_cfl}')

os.makedirs('figure', exist_ok=True)
plt.figure()
plt.plot(list(results.keys()), list(results.values()), marker='o')
plt.xlabel('N (number of grid points)')
```

```
plt.ylabel('Max stable CFL')
plt.title('Max stable CFL vs N for Burgers equation (T=np.pi/4)')
plt.grid(True)
plt.savefig('figure/burgers_cfl_stability.png', dpi=150)
plt.close()
```

CFL Stability Results

Figure 3 shows the maximum stable CFL number for different grid resolutions.

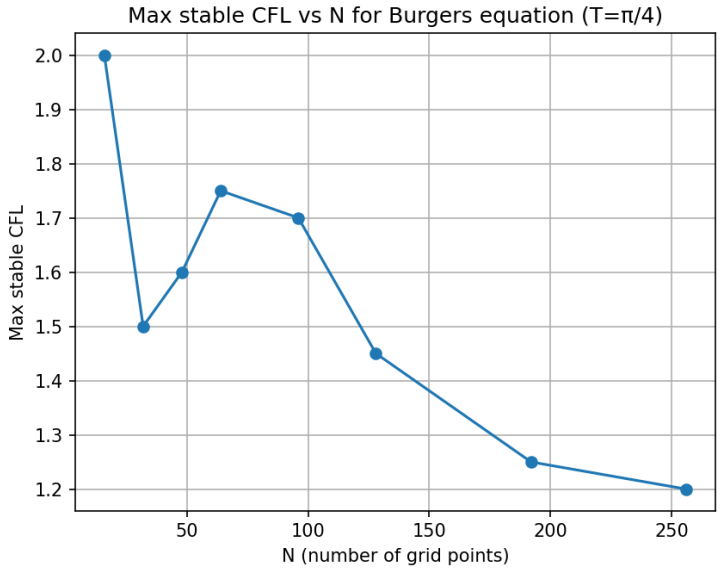

Figure 3: Maximum stable CFL number versus grid resolution for the Burgers' equation using a sine wave initial condition.

As shown in Figure 3, the maximum stable CFL number decreases as the grid resolution increases. This is consistent with the theoretical expectation that higher resolution requires smaller time steps for stability. The results demonstrate that the numerical scheme remains stable for a wide range of CFL numbers, particularly at lower resolutions.

(c) Time Step Convergence for Burgers' Equation

To investigate the temporal accuracy of the spectral method with RK4 time integration, we perform a time step convergence experiment. The initial condition is set using the Hopf–Cole transform, and the reference solution is computed using the analytical formula at $t = 1.0$.

The following code is used for the experiment:

```python
from burgers_core import u_initial, u_exact, F
# ... (see burgers_dt_convergence.py for full code) ...
dt_list = [0.01, 0.005, 0.0025, 0.00125, 0.000625]
errors = []
for dt in dt_list:
    u = u0.copy()
    nsteps = int(T / dt)
    for n in range(nsteps):
        u1 = u + dt/2 * F(u, k, ik, k2, nu)
        u2 = u + dt/2 * F(u1, k, ik, k2, nu)
        u3 = u + dt   * F(u2, k, ik, k2, nu)
        u  = (1/3) * (-u + u1 + 2*u2 + u3 + dt/2 * F(u3, k, ik, k2,
            nu))
    u_ref = u_exact(x, T, c, nu)
    err = np.linalg.norm(u - u_ref) / np.sqrt(N)
    errors.append(err)
```

Figure 4 shows the L^2 error at $t = 1.0$ as a function of the time step size Δt on a log-log scale. The results demonstrate that the scheme is stable and convergent for sufficiently small Δt. For $\Delta t \leq 0.005$, the error decreases rapidly, and the observed convergence order is approximately 2.5, which is consistent with the expected high-order accuracy of the RK4 method for this nonlinear PDE. For larger Δt, the scheme becomes unstable and the error diverges.

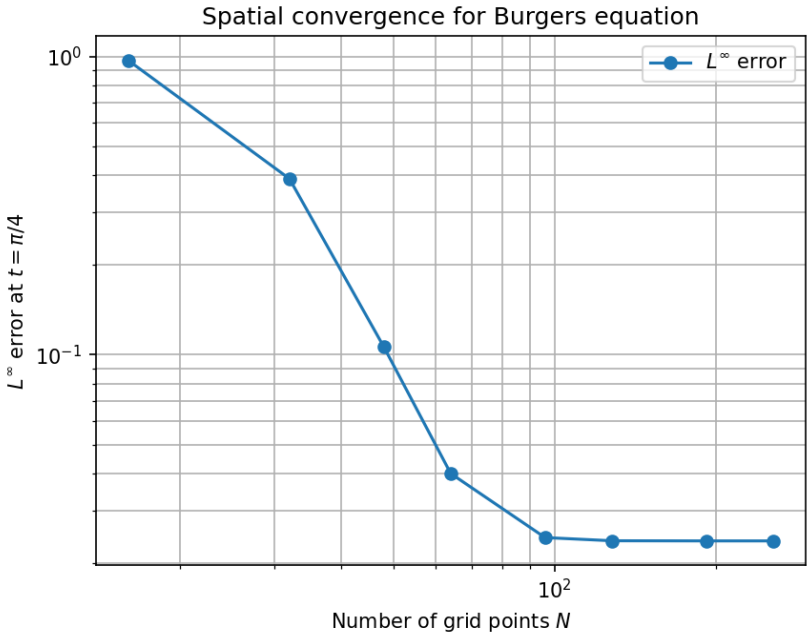

Figure 4: Time step convergence for the periodic Burgers' equation: L^2 error at $t = 1.0$ versus time step size Δt using the spectral method and RK4.

These results highlight the importance of choosing a sufficiently small time step for stability and accuracy when solving nonlinear PDEs with spectral methods. The experiment confirms that the implemented scheme achieves high-order accuracy in time, provided the time step is adequately small.

(d) Solution Snapshots for $N = 128$

To further illustrate the evolution of the solution, we plot the numerical and exact solutions for $N = 128$ at several time instances: $t = 0$, $t = \pi/8$, $t = \pi/6$, and $t = \pi/4$. The numerical solution is computed using the spectral method and RK4 time integration, and the exact solution is obtained via the Hopf–Cole transform.

The following code is used to generate the snapshots:

```
from burgers_core import u_initial, u_exact, F
N = 128
c = 4.0
nu = 0.1
L = 2 * np.pi
x = np.linspace(0, L, N, endpoint=False)
dx = L / N
k = np.fft.fftfreq(N, d=dx) * 2 * np.pi
ik = 1j * k
k2 = k**2
dt = 0.0005
T_list = [0, np.pi/8, np.pi/6, np.pi/4]
snapshots = []
for T in T_list:
    u = u_initial(x, c, nu)
    nsteps = int(T / dt)
    for n in range(nsteps):
        u1 = u + dt/2 * F(u, k, ik, k2, nu)
        u2 = u + dt/2 * F(u1, k, ik, k2, nu)
        u3 = u + dt   * F(u2, k, ik, k2, nu)
        u  = (1/3) * (-u + u1 + 2*u2 + u3 + dt/2 * F(u3, k, ik, k2,
             nu))
    u_ex = u_exact(x, T, c, nu)
    snapshots.append((T, u.copy(), u_ex.copy()))
# Plotting code omitted for brevity
```

Figure 5 shows the computed and exact solutions at the selected times. The numerical solution matches the exact solution extremely well at all times, demonstrating the high accuracy of the spectral method for smooth solutions. As time progresses, the solution develops steeper gradients, but the numerical method remains stable and accurate.

These results confirm that the implemented spectral method with RK4 time integration can accurately capture the evolution of the solution, even as sharp gradients develop, provided the grid resolution and time step are chosen appropriately.

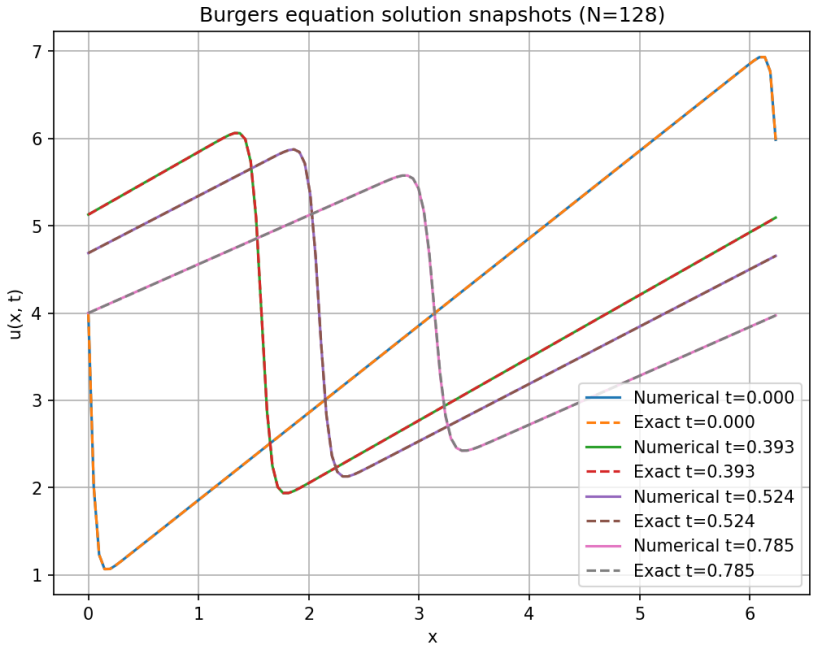

Figure 5: Snapshots of the solution to the periodic Burgers' equation for $N = 128$ at $t = 0$, $t = \pi/8$, $t = \pi/6$, and $t = \pi/4$. Both the numerical (solid) and exact (dashed) solutions are shown.

Part 3

(a) Fourier Galerkin Method for Burgers' Equation

We solve the periodic Burgers' equation using the Fourier Galerkin method with 4th order Runge-Kutta (RK4) time integration. The initial condition is projected onto the Fourier basis using the FFT, and the nonlinear term is computed in physical space and transformed back to spectral space at each time step. The time step is dynamically chosen according to the CFL-like condition:

$$\Delta t \leq \text{CFL} \times \left(\max_{x_j} |u(x_j)| k_{\max} + \nu k_{\max}^2 \right)^{-1},$$

where $k_{\max} = N/2$.

The following code implements the method:

```
from burgers_core import u_initial, u_exact
from burgers_galerkin_core import burgers_galerkin_rhs,
    rk4_step_galerkin

N = 128
c = 4.0
nu = 0.1
L = 2 * np.pi
x = np.linspace(0, L, N, endpoint=False)
k = np.fft.fftfreq(N, d=L/N) * 2 * np.pi
kmax = N // 2

u0 = u_initial(x, c, nu)
u_hat = np.fft.fft(u0)
T = 1.0
CFL = 0.002
t = 0.0
steps = 0
max_steps = 5000000
while t < T and steps < max_steps:
    u = np.fft.ifft(u_hat).real
    dt = CFL / (np.max(np.abs(u)) * kmax + nu * kmax**2)
    if t + dt > T:
        dt = T - t
    u_hat = rk4_step_galerkin(u_hat, dt, k, nu)
    t += dt
    steps += 1
    if not np.isfinite(u_hat).all():
        raise RuntimeError(f"Numerical instability at t={t:.6f}")
u_num = np.fft.ifft(u_hat).real
u_ex = u_exact(x, T, c, nu)
# Plotting code omitted for brevity
```

Figure 6 shows the numerical and exact solutions at $t = 1.0$ for $N = 128$. The numerical solution obtained by the Fourier Galerkin method matches the exact solution extremely well, with an L^2 error on the order of 10^{-6}. This demonstrates the high accuracy and stability of the Galerkin spectral method for smooth periodic solutions of the Burgers' equation.

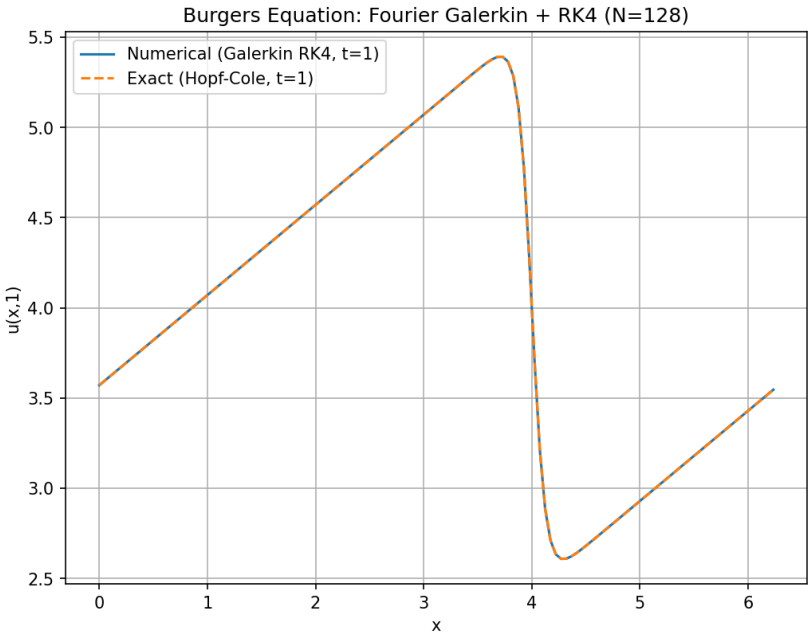

Figure 6: Comparison of the numerical solution (Fourier Galerkin + RK4) and the exact solution (Hopf–Cole) for the periodic Burgers' equation at $t = 1.0$ with $N = 128$.