1 Introduction

Consider the non-linear, inviscid Burgers equation for $u(x,t)$,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0\,, \tag{1}$$

with the initial conditions

$$\begin{aligned} u(x,0) &= 10 \qquad 0 \le x \le 15\,, \\ u(x,0) &= 0 \qquad 30 \le x \le 60\,. \end{aligned} \tag{2}$$

Use the following finite-difference approximations to numerically integrate this equation using appropriate Dirichlet or Neumann BCs on an x-grid with $\Delta x = 0.2$:

1. MacCormack explicit method
2. Beam and Warming implicit method

Note that the second method may require the incorporation of a smoothing operator added directly to the finite difference formula. Using a fourth-order artificial viscosity, optimize the coefficient of this operator for minimum amplitude errors,

$$D_\epsilon = -\epsilon(\Delta x)^4 \frac{\partial^4 u}{\partial x^4}\,, \tag{3}$$

where the negative sign ensures that positive dissipation is produced. Using central differences, we obtain

$$(\Delta x)^4 \frac{\partial^4 u}{\partial x^4} = u_{i-2} - 4u_{i-1} + 6u_i - 4u_{i+1} + u_{i+2}\,. \tag{4}$$

The coefficient ϵ generally obeys $0 \le \epsilon \le 1/8$, with a preferred value of $\epsilon = 0.1$.

For both methods, plot the solutions at intervals of about two time units up to about $t = 8$ time units. Obtain solutions for Courant numbers of $C = \{\frac{3}{4}, 1, \frac{5}{4}\}$. Comment on the stability of the scheme and dispersive/dissipative errors.

2 Methodology

2.1 MacCormack

We can re-write (1) as

$$\frac{\partial u}{\partial t} = -\frac{\partial F}{\partial x}\,, \quad \text{where} \quad F \equiv \frac{u^2}{2}\,. \tag{5}$$

The MacCormack explicit method consists of a predictor and corrector step operating on this equation, using one-sided differences in alternating directions to remove any directional bias from the discretization scheme. The predictor step is

$$\hat{u}_i = u_i^n - \frac{\Delta t}{\Delta x}(F_{i+1}^n - F_i^n)\,, \quad \text{where} \quad F_i^n \equiv \frac{(u_i^n)^2}{2}\,, \tag{6}$$

and the corrector step is

$$u_i^{n+1} = \frac{1}{2}\left[u_i^n + \hat{u}_i - \frac{\Delta t}{\Delta x}(\hat{F}_i^n - \hat{F}_{i-1}^n)\right]\,, \quad \text{where} \quad \hat{F}_i^n \equiv \frac{(\hat{u}_i^n)^2}{2}\,. \tag{7}$$

2.2 Beam and Warming

For the Beam and Warming method, the Crank-Nicolson method advances the solution in time and second-order central differences discretize the solution in space. The resulting finite difference equation is non-linear, and Beam and Warming choose to linearize it rather than iterating to find the non-linear solution. The resulting implicit equation can be written as

$$\overbrace{\left(\frac{-\Delta t}{\Delta x}\frac{u_{i-1}^n}{4}\right)}^{b_i} u_{i-1}^{n+1} + \overbrace{(1)}^{a_i} u_i^{n+1} + \overbrace{\left(\frac{\Delta t}{\Delta x}\frac{u_{i+1}^n}{4}\right)}^{c_i} u_{i+1}^{n+1} = \overbrace{u_i^n - \frac{\Delta t}{2\Delta x}(F_{i+1}^n - F_{i-1}^n) + \frac{\Delta t}{\Delta x}\left((u_{i+1}^n)^2 - (u_{i-1}^n)^2\right) + D_\epsilon}^{d_i}\,, \tag{8}$$

where the coefficients a_i, b_i, c_i, and d_i correspond to diagonal, sub-diagonal, super-diagonal, and right-hand-side terms in a tri-diagonal matrix system, and D_ϵ is defined in (3). The solution vector is obtained via the Thomas algorithm. These concepts have been explained before, and further discussion can be found in Homework 4.

3 Results

MacCormack solutions are presented in Figure 1 for three different Courant numbers.

Beam and Warming results are presented in Figure 2 and Figure 3; the former explores the effect of ϵ on solution behavior, and the latter presents time-dependent results for $\epsilon = 0.1$. All three requested Courant numbers are explored.

4 Discussion

5 References

No external references were used other than the course notes for this assignment.

Appendix: MATLAB Code

The following code listings generate all figures presented in this homework assignment. The Thomas algorithm code is not listed, as it is identical to that used in Homework 4.

Listing 1: `Problem_1.m`

```
function [] = Problem_1()

    %%%%%%
    % Solves the inviscid Burgers equation using the MacCormack explicit method.
    %
    % Ryan Skinner, November 2015
    %%%

    Set_Default_Plot_Properties();

    % For each Courant number...
    for C = [0.75, 1.00, 1.25]

        %%%
        % Define variables specific to the boundary-value problem.
        %%%

        % Solution domain.
        dx = 0.2;
        x_min = 0;
```

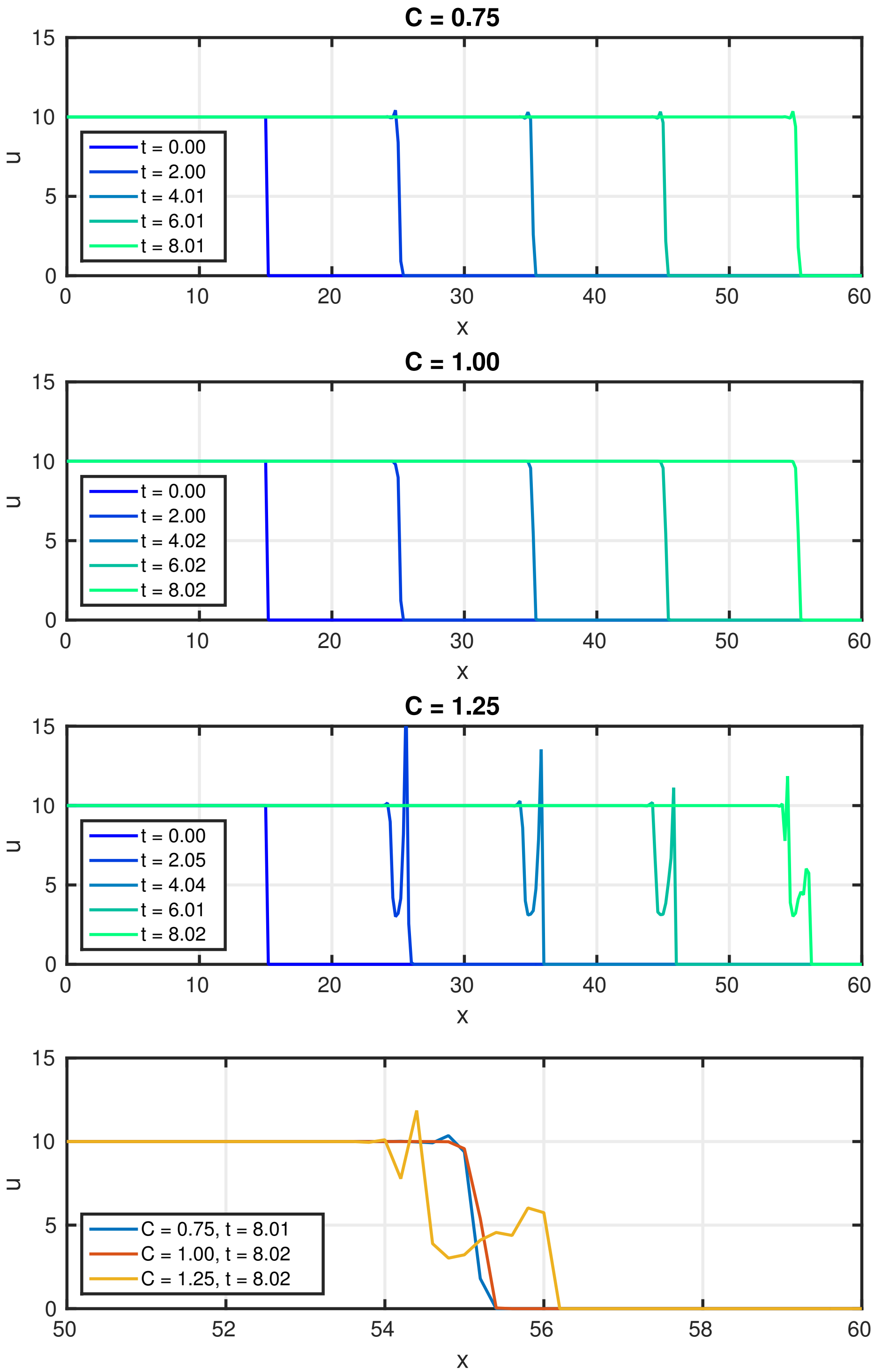

Figure 1: MacCormack method solution for different Courant numbers, and comparison at time $t \sim 8$. The Courant number is seen to substantially affect the accuracy of the results.

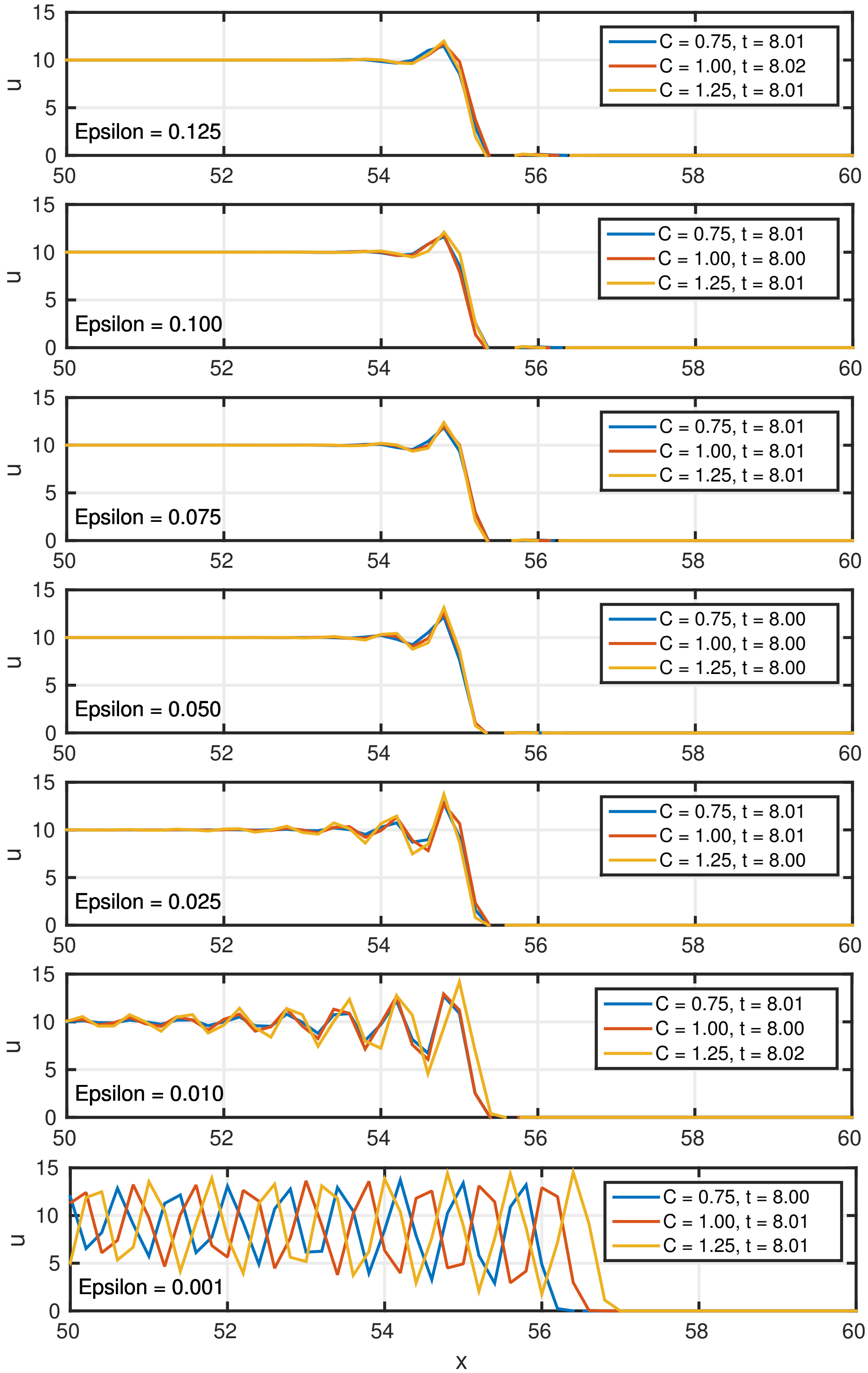

Figure 2: Beam and Warming solution for different values of the dissipation term's coefficient ϵ. Solutions at time $t \sim 8$ are shown for different Courant numbers in each plot. The solution diverged for values of ϵ even a few percent above 0.125. A value of $\epsilon = 0.1$ appears to be sufficient.

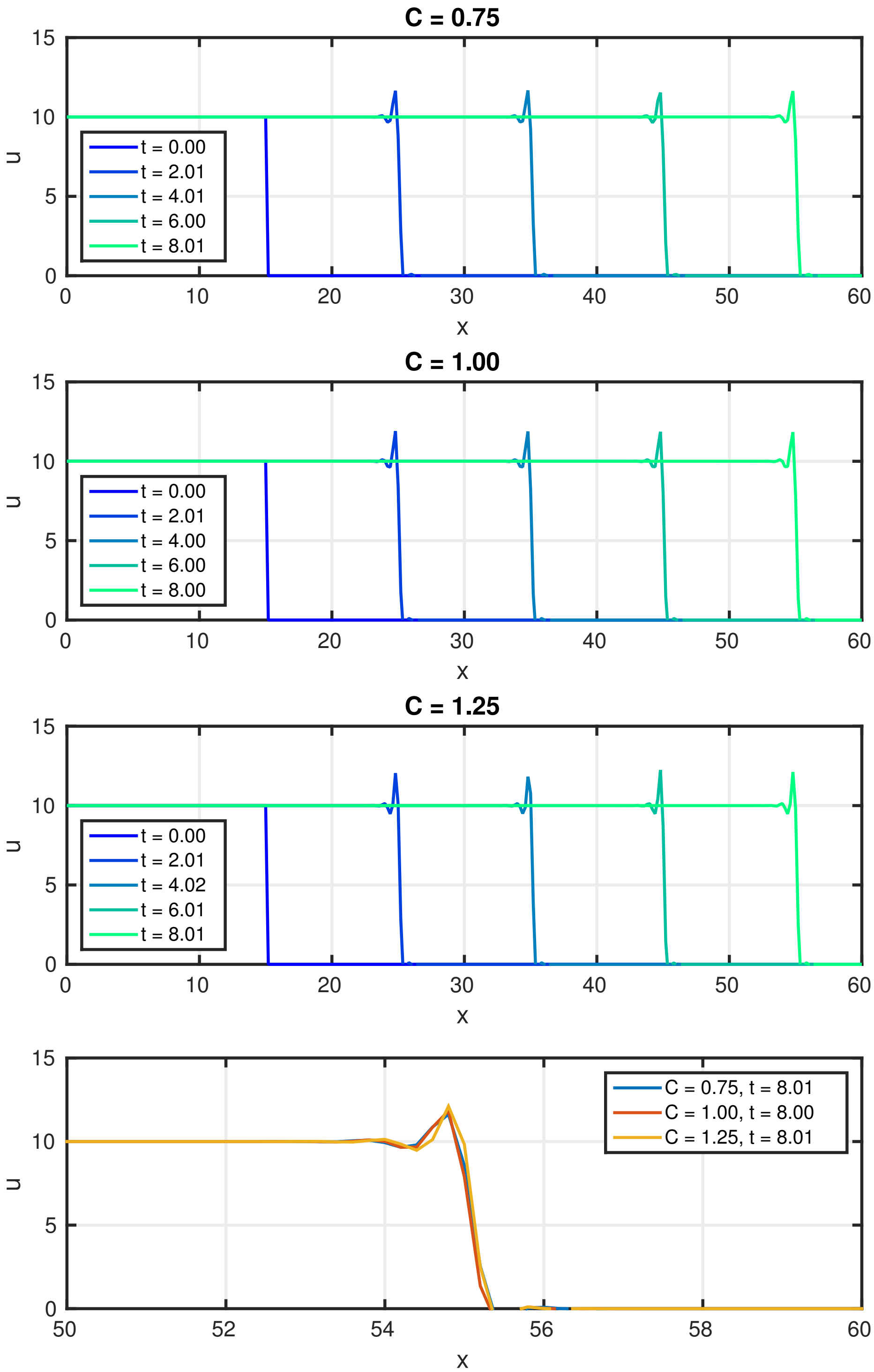

Figure 3: Beam and Warming method solution for three Courant numbers, and comparison at time $t \sim 8$. The dissipation coefficient is $\epsilon = 0.1$. Little difference is observed between different Courant numbers.

```
        x_max = 60;
        x = (x_min:dx:x_max)';
        N = length(x);
        t = 0;
        dt_history = 0;

        % Initialize the solution, indexed by (x,y), and set BCs.
        u = zeros(N,1);
        for i = 1:length(x)
            if x(i) <= 15
                u(i) = 10;
            end
        end

        %%%
        % Solve problem numerically.
        %%%

        % Solve problem up to t~8.
        while t(end) < 8

            u_n = u(:,end);
            F_n = u_n.^2 / 2;
            u_h = nan(N,1);
            u_np1 = nan(N,1);

            % Calculate time step.
            dt = C * dx / max(u_n);
            dt_history(end+1) = dt;
            t(end+1) = t(end) + dt;

            % MacCormack iterations.

            calc_u_hat = @(uni,Fnip1,Fni)       uni                                  ...
                                                    - (dt/dx) * (Fnip1 - Fni);
            calc_u_np1 = @(uni,uhi,Fhip1,Fhi) ( uni + uhi                            ...
                                                    - (dt/dx) * (Fhip1 - Fhi) ) / 2;

            for i = 1:N-1
                u_h(i) = calc_u_hat(u_n(i), F_n(i+1), F_n(i));
            end
            u_h(N) = u_h(N-1);
            F_h = u_h.^2 / 2;

            for i = 2:N
                u_np1(i) = calc_u_np1(u_n(i), u_h(i), F_h(i), F_h(i-1));
            end
            u_np1(1) = u_np1(2);

            % Update solution.
            u(:,end+1) = u_np1;

        end

        %%%
        % Process results.
        %%%

        % Time evolution of solution for a single Courant number.
        n_plot = 5;
        cmap = winter(n_plot);
        step_numbers = round(linspace(1,length(t),n_plot));
        hf = figure(round(C*10));
        set(hf,'Position',[100,500,900,300]);
        hold on;
        for i = 1:length(step_numbers)
            tmp = sprintf('t = %.2f', t(step_numbers(i)));
            plot(x, u(:,step_numbers(i)), 'DisplayName', tmp, 'Color', cmap(i,:));
        end
```

```
        title(sprintf('C = %.2f',C));
        xlabel('x');
        ylabel('u');
        ylim([0,15]);
        xlim([x_min,x_max]);
        hleg = legend('show');
        set(hleg,'Location','southwest');

        % Solution at t~8, comparing Courant numbers.
        hf = figure(1);
        set(hf,'Position',[100,500,900,300]);
        hold on;
        tmp = sprintf('C = %.2f, t = %.2f', C, t(end));
        plot(x, u(:,end), 'DisplayName', tmp);
        xlabel('x');
        ylabel('u');
        ylim([0,15]);
        xlim([50,60]);

%         figure();
%         surf(x,t,u');
%         xlabel('x');
%         ylabel('t');
%         title(sprintf('C = %.2f',C));
%         xlim([x_min,x_max]);
%         ylim([min(t),max(t)]);
%         zlim([0,15]);

    end

    figure(1);
    hleg = legend('show');
    set(hleg,'Location','southwest');

    disp('Done.');
    return

end
```

Listing 2: `Problem_2.m`

```
function [] = Problem_2()

    %%%%%%
    % Solves the inviscid Burgers equation using the Beam and Warming implicit method.
    %
    % Ryan Skinner, November 2015
    %%%

    Set_Default_Plot_Properties();

    clear

    % For each Courant number...
    for C = [0.75, 1.00, 1.25]
        epsilon = 0.1;

        %%%
        % Define variables specific to the boundary-value problem.
        %%%

        % Solution domain.
        dx = 0.2;
        x_min = 0;
        x_max = 60;
        x = (x_min:dx:x_max)';
        N = length(x);
        t = 0;

```

```
        % Initialize the solution, indexed by (x,y), and set BCs.
        u = zeros(N,1);
        for i = 1:length(x)
            if x(i) <= 15
                u(i) = 10;
            end
        end

        %%%
        % Solve problem numerically.
        %%%

        % Solve problem up to t~8.
        while t(end) < 8

            u_n = u(:,end);
            dt = C * dx / max(u_n);
            t(end+1) = t(end) + dt;

            % Beam and Warming implicit iteration.
            [diag, sub, sup, rhs] = Assemble_BeamWarming(u_n, epsilon, dt, dx);
            [sol] = Thomas(diag, sub, sup, rhs);
            u_np1 = [10; sol; 0];
            u_np1(1:10) = 10;

            % Update solution.
            u(:,end+1) = u_np1;

        end

        %%%
        % Process results.
        %%%

        % Time evolution of solution for a single Courant number.
        n_plot = 5;
        cmap = winter(n_plot);
        step_numbers = round(linspace(1,length(t),n_plot));
        hf = figure(round(C*10));
        set(hf,'Position',[100,500,900,300]);
        hold on;
        for i = 1:length(step_numbers)
            tmp = sprintf('t = %.2f', t(step_numbers(i)));
            plot(x, u(:,step_numbers(i)), 'DisplayName', tmp, 'Color', cmap(i,:));
        end
        title(sprintf('C = %.2f',C));
        xlabel('x');
        ylabel('u');
        ylim([0,15]);
        xlim([x_min,x_max]);
        hleg = legend('show');
        set(hleg,'Location','southwest');

        % Solution at t~8, comparing Courant numbers.
        hf = figure(1);
        set(hf,'Position',[100,500,900,300]);
        hold on;
        tmp = sprintf('C = %.2f, t = %.2f', C, t(end));
        plot(x, u(:,end), 'DisplayName', tmp);
        text(50.1,2.5,sprintf('Epsilon = %.3f', epsilon),'FontSize',14);
        xlabel('x');
        ylabel('u');
        ylim([0,15]);
        xlim([50,60]);

    end

    figure(1);
    hleg = legend('show');
```

```
    set(hleg,'Location','northeast');

    disp('Done.');
    return

end
```

Listing 3: `Assemble_BeamWarming.m`

```
function [diag, sub, sup, rhs] = Assemble_BeamWarming( u, epsilon, dt, dx )

    %%%%%%
    % Assembles the LHS matrix and the RHS vector for the Beam and Warming system
    %   diag -- diagonal
    %    sub -- sub-diagonal
    %    sup -- super-diagonal
    %    rhs -- right-hand side vector
    %
    % Ryan Skinner, November 2015
    %%%

    F = u.^2 / 2;

    N = length(u);

    diag_range = 2:N-1;
     sub_range = 3:N-1;
     sup_range = 2:N-2;

    diag = ones(length(diag_range),1);
     sub = - (1/4) * (dt/dx) *  u(sub_range-1);
     sup =   (1/4) * (dt/dx) *  u(sup_range+1);
     rhs = u(diag_range) ...
          - (1/2) * (dt/dx) * (F(diag_range+1) - F(diag_range-1)) ...
          + (1/4) * (dt/dx) *  u(diag_range+1).^2 ...
          - (1/4) * (dt/dx) *  u(diag_range-1).^2 ;

    % Account for boundaries.
    rhs(1)   = rhs(1)   + (1/4) * (dt/dx) * 10;
    rhs(end) = rhs(end) - (1/4) * (dt/dx) *  0;

    % Add the artificial viscosity.
    De = zeros(length(diag_range),1);
    for i = 1:length(diag_range)
        ii = i+1;

        if ii-2>0 tmp=u(ii-2);      else tmp=10; end
        De(i) = De(i) + tmp;

        if ii-1>0 tmp = -4*u(ii-1); else tmp=-4*10; end
        De(i) = De(i) + tmp;

        De(i) = De(i) + 6 * u(ii);

        if ii+1<N+1 tmp=-4*u(ii+1); else tmp=0; end
        De(i) = De(i) + tmp;

        if ii+2<N+1 tmp=u(ii+2);    else tmp=0; end
        De(i) = De(i) + tmp;
    end
    De = -epsilon * De;
    rhs = rhs + De;

end
```