Caleb Logemann
MATH667 Hyperbolic Partial Differential Equations
Homework 6

1. (a) The following are my methods for the second order Upwind, Central and MUSCL schemes. The main difference in these functions is the different definitions of the numerical flux.

```
function [u] = upwindFV2(f, u0, deltaT, deltaX, nTimeSteps)
    nGridCells = length(u0);
    u = zeros(nTimeSteps+1, nGridCells);
    u(1, :) = u0;
    nu = deltaT/deltaX;

    boundaryConditions = 'periodic';

    % flux array, F(i) is flux at i + 1/2 interface
    F = zeros(nGridCells,1);

    for n = 1:nTimeSteps
        % compute fluxes at boundaries
        for j = 1:nGridCells
            % boundary conditions
            jm1 = j-1;
            if (j == 1)
                if (strcmp(boundaryConditions,'periodic'))
                    jm1 = nGridCells;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jm1 = 1;
                end
            end
            F(j) = f(1.5*u(n,j) - 0.5*u(n,jm1));
        end

        % update solution
        for j = 1:nGridCells
            % boundary conditions
            jm1 = j-1;
            if (j == 1)
                if (strcmp(boundaryConditions,'periodic'))
                    jm1 = nGridCells;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jm1 = 1;
                end
            end

            u(n+1, j) = u(n, j) + nu*(F(jm1) - F(j));
        end
    end
end
```

```
function [u] = centralFV2(f, u0, deltaT, deltaX, nTimeSteps)
    nGridCells = length(u0);
    u = zeros(nTimeSteps+1, nGridCells);
    u(1, :) = u0;
    nu = deltaT/deltaX;

    boundaryConditions = 'periodic';
```

```
    % flux array, F(i) is flux at i - 1/2 interface
    F = zeros(nGridCells,1);

    for n = 1:nTimeSteps
        % compute fluxes at boundaries
        for j = 1:nGridCells
            % boundary conditions
            jm1 = j-1;
            if (j == 1)
                if (strcmp(boundaryConditions,'periodic'))
                    jm1 = nGridCells;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jm1 = 1;
                end
            end
            F(j) = f(0.5*(u(n,jm1) + u(n,j)));
        end

        % update solution
        for j = 1:nGridCells
            % boundary conditions
            jp1 = j+1;
            if (j == nGridCells)
                if (strcmp(boundaryConditions,'periodic'))
                    jp1 = 1;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jp1 = nGridCells;
                end
            end

            u(n+1, j) = u(n, j) + nu*(F(j) - F(jp1));
        end
    end
end
```

```
function [u] = muscl2(f, u0, deltaT, deltaX, nTimeSteps)
    nGridCells = length(u0);
    u = zeros(nTimeSteps+1, nGridCells);
    u(1, :) = u0;
    nu = deltaT/deltaX;

    boundaryConditions = 'periodic';

    % flux array, F(i) is flux at i + 1/2 interface
    F = zeros(nGridCells,1);

    for n = 1:nTimeSteps
        % compute fluxes at boundaries
        for j = 1:nGridCells
            % boundary conditions
            jm1 = j-1;
            if (j == 1)
                if (strcmp(boundaryConditions,'periodic'))
                    jm1 = nGridCells;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jm1 = 1;
                end
            end

            jp1 = j+1;
```

```
            if (j == nGridCells)
                if (strcmp(boundaryConditions,'periodic'))
                    jp1 = 1;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jp1 = nGridCells;
                end
            end
            % upwind
            u1 = 1.5*u(n,j) - 0.5*u(n,jm1);
            % central
            u2 = 0.5*(u(n,j) + u(n,jp1));
            F(j) = f(minmod(u1, u2));
        end

        % update solution
        for j = 1:nGridCells
            % boundary conditions
            jm1 = j-1;
            if (j == 1)
                if (strcmp(boundaryConditions,'periodic'))
                    jm1 = nGridCells;
                elseif (strcmp(boundaryConditions,'zeroFlux'))
                    jm1 = 1;
                end
            end

            u(n+1, j) = u(n, j) + nu*(F(jm1) - F(j));
        end
    end
end
```

The following script now performs a test for the order of accuracy of these three methods.

```
%% Problem 1 (a)
u0func = @(x) 1 + 0.5*sin(x);
Iu0func = @(x) x - 0.5*cos(x);
du0func = @(x) 0.5*cos(x);
a = 0;
b = 2*pi;
tFinal = 1.0;
f = @(u) (u^2)/2;

for method = ["centralFV2", "upwindFV2", "muscl2"]
    E = zeros(4, 4);
    iter = 0;

    for N = [20, 40, 80, 160]
        iter = iter + 1;
        deltaX = (b - a)/N;
        x = linspace(a+0.5*deltaX, b-0.5*deltaX, N);
        u0 = zeros(N,1);
        for i = 1:N
            u0(i) = (1/deltaX)*(Iu0func(x(i) + 0.5*deltaX) - Iu0func(x(i) - 0.5*
                ↪ deltaX));
        end

        deltaT = 0.25*deltaX;
        nTimeSteps = ceil(tFinal/deltaT);
        deltaT = tFinal/nTimeSteps;
```

```
        sol = feval(char(method), f, u0, deltaT, deltaX, nTimeSteps);
        exactSol = burgersExactSolution(x, u0func, du0func, tFinal);
        plot(x, sol(end,:), x, exactSol);
        pause();


        E(iter, 1) = N;
        E(iter, 2) = deltaX;
        E(iter, 3) = max(abs(sol(end,:) - exactSol'));
        if(iter >= 2)
            E(iter, 4) = log(E(iter-1, 3)/E(iter, 3))/log(E(iter-1, 2)/E(iter, 2));
        end
    end
    disp(latexFileWriter.printMatrix(E,3));
end
```

Note that I am using the forward Euler timestepping, so the methods should only achieve first order accuracy even though the methods are second order in space.

Central Scheme

N	Δx	L^∞ Error	Order
20	0.314	0.075	-
40	0.157	0.035	1.103
80	0.079	0.017	1.059
160	0.039	0.008	1.085

Upwind Scheme

N	Δx	L^∞ Error	Order
20	0.314	0.038	-
40	0.157	0.023	0.745
80	0.079	0.013	0.846
160	0.039	0.007	0.961

MUSCL Scheme

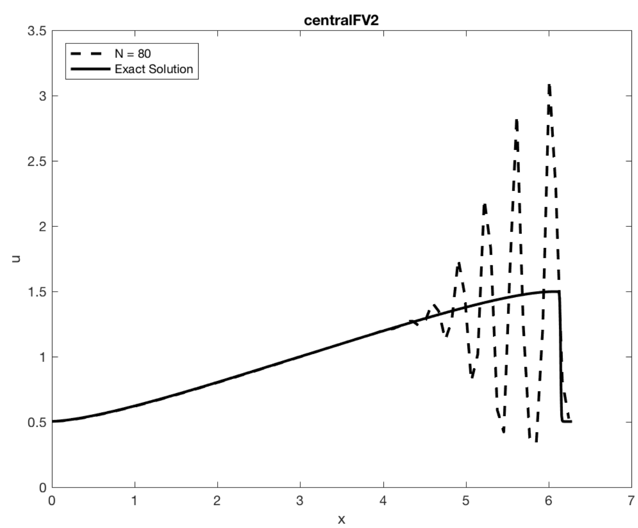
N	Δx	L^∞ Error	Order
20	0.314	0.067	-
40	0.157	0.031	1.098
80	0.079	0.015	1.081
160	0.039	0.007	1.018

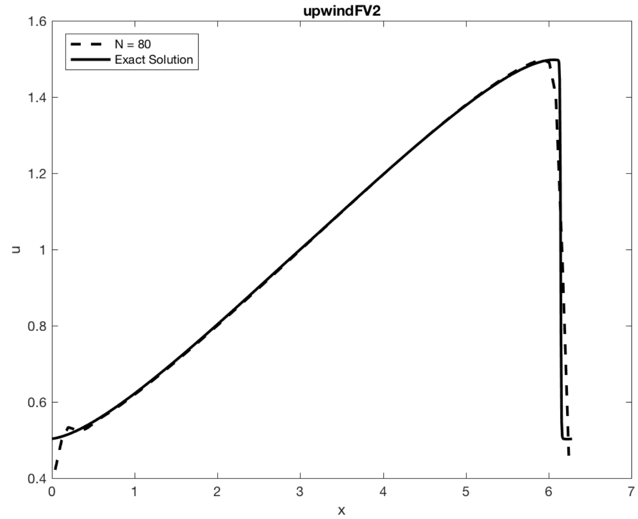
(b) The following script now uses the same methods in part (a) to simulate out to $t = 3.0$.

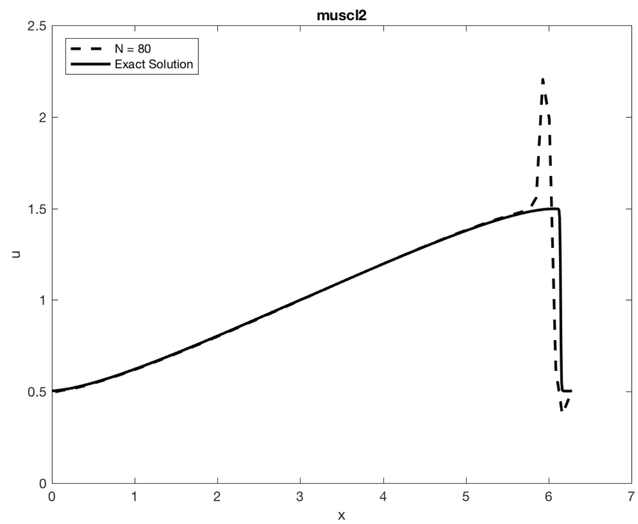
```
%% Problem 1 (b)
u0func = @(x) 1 + 0.5*sin(x);
Iu0func = @(x) x - 0.5*cos(x);
a = 0;
b = 2*pi;
tFinal = 3.0;
f = @(u) (u^2)/2;
style = ["k--", "k-"];

% exact solution
N = 800;
deltaX = (b - a)/N;
```

```
xExact = linspace(a, b, N);
u0 = u0func(xExact);

deltaT = 0.5*deltaX;
nTimeSteps = ceil(tFinal/deltaT);
deltaT = tFinal/nTimeSteps;
exactSol = godunov(f, u0, deltaT, deltaX, nTimeSteps);

iter1 = 0;
for method = ["centralFV2", "upwindFV2", "muscl2"]
    iter1 = iter1+1;
    iter = 0;
    N = 80;
    iter = iter+1;
    deltaX = (b - a)/N;
    x = linspace(a+0.5*deltaX, b-0.5*deltaX, N);
    u0 = zeros(N,1);
    for i = 1:N
        u0(i) = (1/deltaX)*(Iu0func(x(i) + 0.5*deltaX) - Iu0func(x(i) - 0.5*deltaX)
            ↪ );
    end

    deltaT = 0.1*deltaX;
    nTimeSteps = ceil(tFinal/deltaT);
    deltaT = tFinal/nTimeSteps;

    sol = feval(char(method), f, u0, deltaT, deltaX, nTimeSteps);
    plot(x, sol(end,:), 'k--', xExact, exactSol(end,:), 'k-', 'LineWidth', 2);
    xlabel('x');
    ylabel('u');
    title(char(method));
    legend('N = 80', 'Exact Solution', 'Location', 'northwest');
    saveas(gcf, strcat('Figures/06_0',num2str(iter1),'.png'), 'png');
end
```

The following three images are produced. The exact solution is computed using Godunov's method, as these methods are still inaccurate at $N = 800$.







2. The following script uses the methods from problem 1 on the advection equation.

```
%% Problem 2
%u0func = @(x) exp(-x.^2);
%Iu0func = @(x) sqrt(pi)/2*erf(x);
u0func = @(x) (x.^2)*(-1 < x && x < 1);
Iu0func = @(x) (1/3*x.^3)*(-1 < x && x < 1);
a = -pi;
b = pi;
f = @(u) u;
exactSol = @(x, t) u0func(x - t);
style = ["k--", "k-"];
iter1 = 0;
for tFinal = [1.0, 2.0]
    for method = ["upwindFV2", "muscl2"]
        iter1 = iter1+1;
        figure;
        hold on;
        N = 80;
        iter = iter+1;
        deltaX = (b - a)/N;
```

```
        x = linspace(a+0.5*deltaX, b-0.5*deltaX, N);
        u0 = zeros(N,1);
        for i = 1:N
            u0(i) = (1/deltaX)*(Iu0func(x(i) + 0.5*deltaX) - Iu0func(x(i) - 0.5*deltaX));
        end

        deltaT = 0.01*deltaX;
        nTimeSteps = ceil(tFinal/deltaT);
        deltaT = tFinal/nTimeSteps;

        sol = feval(char(method),f, u0, deltaT, deltaX, nTimeSteps);
        plot(x, sol(end,:), 'k--', 'LineWidth', 2);
        exactSolution = zeros(N,1);
        for i = 1:N
            exactSolution(i) = exactSol(x(i), tFinal);
        end
        plot(x, exactSolution, 'k-', 'LineWidth', 2);
        xlabel('x');
        ylabel('u');
        title(strcat(char(method),' at t = ', num2str(tFinal)));
        legend('N = 80', 'Exact Solution', 'Location', 'northwest');
        hold off;
        saveas(gcf, strcat('Figures/06_0',num2str(iter1+3),'.png'), 'png');
    end
end
```

The following images are produced. Again these are using 2nd order space discretizations without any slope limiters, so oscillations occur around discontinuities.