Caleb Logemann
AER E 546 Fluid Mechanics and Heat Transfer I
Homework 3

1. A slab of metal is initially at uniform temperature. One end is suddenly raised to a high temperature, while the other end is kept cool. Compute the penetration of heat into the slab as a function of time. In dimensional form the temperature diffusion equation, initial and boundary values are

$$\frac{\partial T^*}{\partial t_*} = \kappa \frac{\partial^2 T^*}{\partial {x_*}^2} \qquad T^*(x,0) = 0 \qquad T^*(0,t) = 0, T^*(L,t) = T_w.$$

Non-dimensionalize temperature by T_w and length by L and time by L^2/κ. Integrate by Euler Explicit, up to a non-dimensional time of 0.3. Use $N_x = 121$ grid points in x. Let $\Delta t = \alpha \Delta x^2$. Try a value of $\alpha > 0.5$. What happens? Why? How small must α be to obtain an accurate solution? Provide a single figure with line plots of the solution at time intervals of 0.04. Note that the computational time-step will be smaller than 0.04. The bulk heat transfer coefficient is defined as

$$h_T = \frac{Q}{T(1) - T(0)}$$

where $T = \left.\frac{\partial T}{\partial x}\right|_{x=1}$ is the heat flux into the slab. Plot h_T as a function of time for $t > 0.01$.

First I will nondimensionalize this equation by doing the following substitutions.

$$T^* = T_w T \tag{1}$$
$$x^* = Lx \tag{2}$$
$$t_* = \frac{L^2}{\kappa} t \tag{3}$$

Now the differential equation can be simplified as follows.

$$\frac{\partial T^*}{\partial t_*} = \kappa \frac{\partial^2 T^*}{\partial {x_*}^2} \tag{4}$$
$$T_w \frac{\partial T}{\partial t_*} = \kappa T_w \frac{\partial^2 T}{\partial {x_*}^2} \tag{5}$$
$$\frac{T_w \kappa}{L^2} \frac{\partial T}{\partial t} = \frac{\kappa T_w}{L^2} \frac{\partial^2 T}{\partial x^2} \tag{6}$$
$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \tag{7}$$

The initial and boundary conditions are given as

$$T(x,0) = 0 \qquad T(0,t) = 0 \qquad T(1,t) = T_2$$

Now this non-dimensionalized PDE can be descretized in space as follows

$$\dot{T}_j = \frac{T_{j+1} - 2T_j + T_{j-1}}{\Delta x^2}$$

If the time derivative is solved with Euler Explicit then the update formula becomes

$$T_j^{n+1} = T_j^n + \frac{\Delta t}{\Delta x^2}(T_{j+1} - 2T_j + T_{j-1})$$

The following function implements this Euler Explicit update.

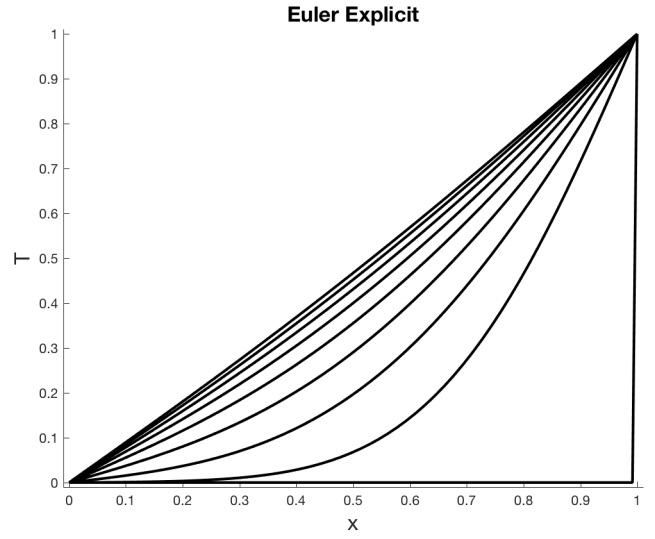
```
function [result] = forwardEuler(RHSFunc, x0, nTimeSteps, deltaT)
    n = length(x0);
    result = zeros(nTimeSteps+1, n);
    result(1,:) = x0;

    for i = 1:nTimeSteps
        t = (i-1)*deltaT;
        result(i+1, :) = result(i,:) + deltaT*RHSFunc(t, result(i,:));
    end
end
```

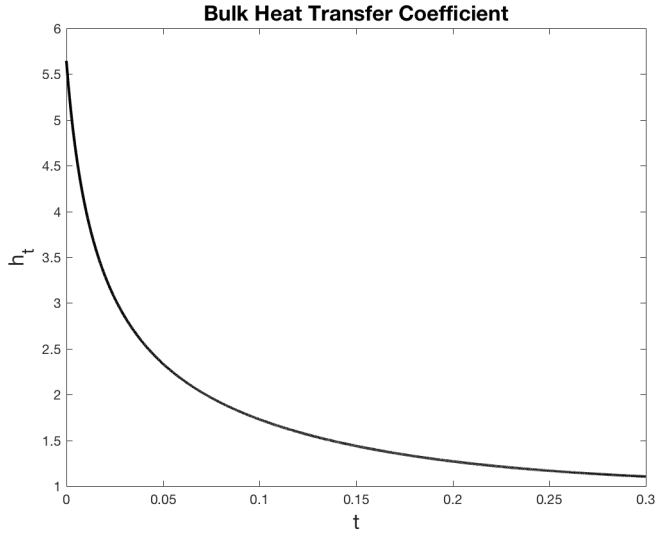
The following script now uses the forwardEuler function to compute a solution to the non-dimensional problem.

```
%% Problem 1
n = 121;
deltaX = 1/n;
x = linspace(deltaX, 1-deltaX, n-1);
tFinal = 0.3;
alpha = 0.5;
deltaT = alpha*deltaX^2;
nTimeSteps = ceil(tFinal/deltaT);
deltaT = tFinal/nTimeSteps;

% initial conditions
T0Func = @(x) zeros(size(x));
T0 = T0Func(x);

% Boundary condtions
Tl = 0;
Tr = 1;
RHSFunc = @(t, T) ([T(2:end),Tr]-2*T+[Tl,T(1:end-1)])/deltaX^2;

sol = forwardEuler(RHSFunc, T0, nTimeSteps, deltaT);

plotInterval = 0.04;
nPlots = ceil(tFinal/plotInterval);
timeStepInterval = floor(nTimeSteps/nPlots);
figure;
hold on
for n = 1:timeStepInterval:nTimeSteps
    plot([0,x,1], [0,sol(n,:),1], 'k', 'LineWidth', 2);
end
hold off
xlim([-0.01,1.01]);
ylim([-0.01,1.01]);
xlabel('x', 'FontSize', 16);
ylabel('T', 'FontSize', 16);
title('Euler Explicit', 'FontSize', 16);
saveas(gcf, 'Figures/03_01.png', 'png');

tStart = 0.01;
nInitial = ceil(tStart/deltaT);
ht = (ones(nTimeSteps - nInitial+2,1) - sol(nInitial:end,end))/deltaX;
t = linspace(0,tFinal, nTimeSteps - nInitial+2);
plot(t,ht, 'k', 'LineWidth', 2);
xlabel('t', 'FontSize', 16);
ylabel('h_t', 'FontSize', 16);
title('Bulk Heat Transfer Coefficient', 'FontSize', 16);
```

```
saveas(gcf, 'Figures/03_02.png', 'png');
```

The script outputs the following two images.





2. A slab is heated by shining a laser on it. The laser is shut off and the heat diffuses throughout the slab. Its ends are insulated. This is modeled as the non-dimensional problem: solve

$$\frac{\partial T}{\partial x} = \frac{\partial}{\partial x}\left(\kappa \frac{\partial T}{\partial x}\right)$$

in the interval $0 \leq x \leq 1$, with the initial condition

$$T(x,0) = \frac{e^{-(x-0.5)^2/\sigma^2}}{\sigma\sqrt{\pi}}, \quad \sigma = 0.1$$

The slab length is normalized to unity and σ characterizes the region heated by the laser. Consider a material with variable diffusivity. Let

$$\kappa = 0.1 + 2.0e^{-5x}$$

The no-flux boundary condition

$$\frac{\partial T}{\partial x}(0,t) = 0 = \frac{\partial T}{\partial x}(1,t)$$

is applied at the insulated ends. Use second order Runge-Kutta. Solve with about $N_x = 250$ grid points in x. Choose a small time-step to obtain an accurate solution. Integrate up to a non-dimensional time of 0.05. Provide a single plot containing the initial condition and curves showing the solution $T(x)$ at time intervals of 0.01. Plot $\int_0^1 T(x)\,\mathrm{d}x$ versus time. What should the value of the integral be?

Since the slab has a variable diffusivity, this problem can not be discretized in space the same way as before. I chose to rewrite the problem as

$$\frac{\partial T}{\partial t} = \kappa'(x)\frac{\partial T}{\partial x} + \kappa(x)\frac{\partial^2 T}{\partial x^2}$$

I then use the standard central finite difference approximations for the first and second space derivatives. This results in the following set of ODEs

$$\dot{T}_j = \kappa'(x_j)\frac{T_{j+1} - T_{j-1}}{2\Delta x} + \kappa(x_j)\frac{T_{j+1} - 2T_j + T_{j-1}}{\Delta x^2}$$

The exact value of the derivative of κ can be used.

$$\kappa'(x) = -10.0e^{-5x}$$

This system of ODEs can now be solved using RK2. The following function implements RK2.

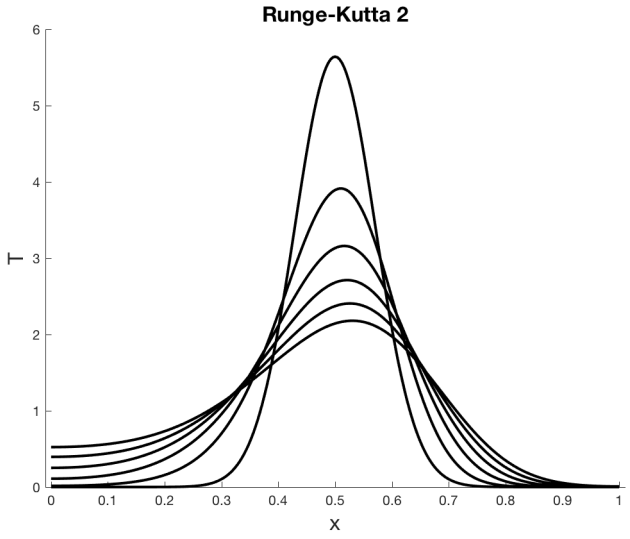
```
function [result] = RK2(RHSFunc, x0, nTimeSteps, deltaT)
    n = length(x0);
    result = zeros(nTimeSteps+1, n);
    result(1,:) = x0;

    for i = 1:nTimeSteps
        t = (i-1)*deltaT;
        temp = result(i,:) + 0.5*deltaT*RHSFunc(t, result(i,:));
        result(i+1, :) = result(i,:) + deltaT*RHSFunc(t + 1/2*deltaT, temp);
    end
end
```

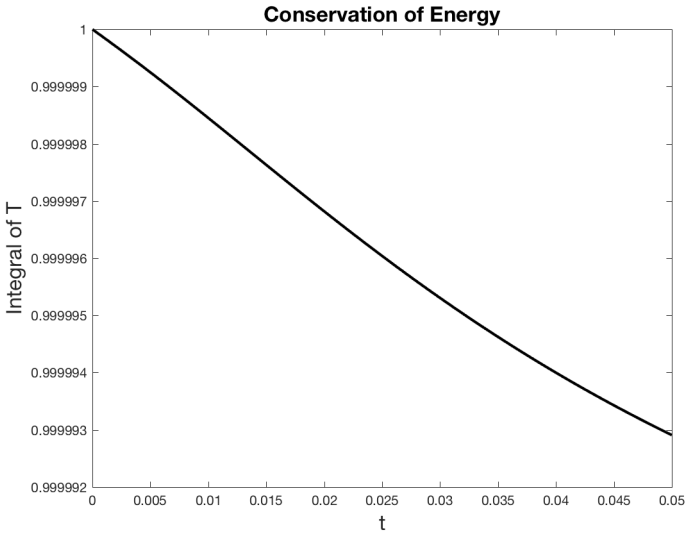
The following script now uses this function to solve the system of ODEs given earlier.

```
%% Problem 2
n = 250;
tFinal = 0.05;
a = 0;
b = 1;
deltaX = (b - a)/n;
x = linspace(a, b, n+1);
deltaT = 0.1*deltaX^2;
nTimeSteps = ceil(tFinal/deltaT);
deltaT = tFinal/nTimeSteps;
kappa = @(x) 0.1 + 2.0*exp(-5*x);
dkappa = @(x) -10.0*exp(-5*x);

% Initial conditions
sigma = 0.1;
T0Func = @(x) exp(-(x - 0.5).^2/sigma^2)/(sigma*sqrt(pi));
```

```
T0 = T0Func(x);

% Boundary Conditions
% zero flux
RHSFunc = @(t, T) dkappa(x).*(([T(2:end), T(end)] - [T(1), T(1:end-1)])/(2*deltaX)) ...
    + kappa(x).*(([T(2:end),T(end)]-2*T+[T(1),T(1:end-1)])/deltaX^2);

sol = RK2(RHSFunc, T0, nTimeSteps, deltaT);

plotInterval = 0.01;
nPlots = ceil(tFinal/plotInterval);
timeStepInterval = floor(nTimeSteps/nPlots);
figure;
hold on
for n = 1:timeStepInterval:nTimeSteps
    plot(x, sol(n,:), 'k', 'LineWidth', 2);
end
hold off
xlim([-0.01,1.01]);
%ylim([-0.01,1.01]);
xlabel('x', 'FontSize', 16);
ylabel('T', 'FontSize', 16);
title('Runge-Kutta 2', 'Fontsize', 16);
saveas(gcf, 'Figures/03_03.png', 'png');

intT = sum(deltaX*sol, 2);
t = linspace(0, tFinal, nTimeSteps+1);
plot(t, intT, 'k', 'LineWidth', 2);
xlabel('t', 'FontSize', 16);
ylabel('Integral of T', 'FontSize', 16);
title('Conservation of Energy', 'FontSize', 16);
saveas(gcf, 'Figures/03_04.png', 'png');
```

The script outputs the following two images. The first image shows the temerature distribution over time. The second image shows the integral of the temperature over time. Note that since both ends of the slab are insulated no energy should be leaving the slab, so this plot should be constant at 1. This is not exactly true, but the total integral only decreases by about 0.000008, so energy is almost exactly conserved. It is also good to note that this is a decrease in energy, no extra heat is being generated.





3. Now consider the case where one end of the slab is insulated and the other is held at constant temperature:

$$\frac{\partial T}{\partial x}(0) = 0 \quad T(1) = 1$$

solve the constant diffusivity, diffusion equation as in the first problem, but use Crank-Nicholson.

(a) Set $\Delta t = \alpha \Delta x^2$. Try a couple of relatively large value of α and see whether your calculation converges, or blows up (should it?).

Using some large values of α, like $\alpha = 10$, I see that the solution does converge. It doesn't blow up, I do see some strange behavior in the bulk heat transfer coefficient at the beginning of the simulation, but the numerical solution still converges to the correct solution. This is because the Crank-Nicholson method is absolutely stable for any time-step.

(b) Provide a single figure with plots of $T(x)$ at intervalue of 0.04 up to $t = 0.4$. Explain why your solution makes sense. The bulk heat transfer coefficient is defined as

$$h_T = \frac{Q}{T(1) - T(0)}$$

where $T = \left.\frac{\partial T}{\partial x}\right|_{x=1}$ is the heat flux into the slab. Plot h_T as a function of time for $t > 0.01$.

The following function implements the Crank-Nicholson method.

```
function [result] = crankNicholson(x0, nTimeSteps, deltaT, deltaX)
    n = length(x0);
    result = zeros(nTimeSteps+1, n);
    result(1,:) = x0;
    a = 2*deltaX^2/deltaT;

    mainDiagonal = -(a + 2)*ones(n, 1);
    lowerDiagonal = ones(n-1,1);
    upperDiagonal = ones(n-1,1);
    % zero flux on left boundary
    mainDiagonal(1) = -(a + 2) + 1;

    for i = 1:nTimeSteps
        RHS = -[result(i,2:end),1] - (a - 2)*result(i,:) - [result(i,1), result(i,1
            ↪ :end-1)];
```

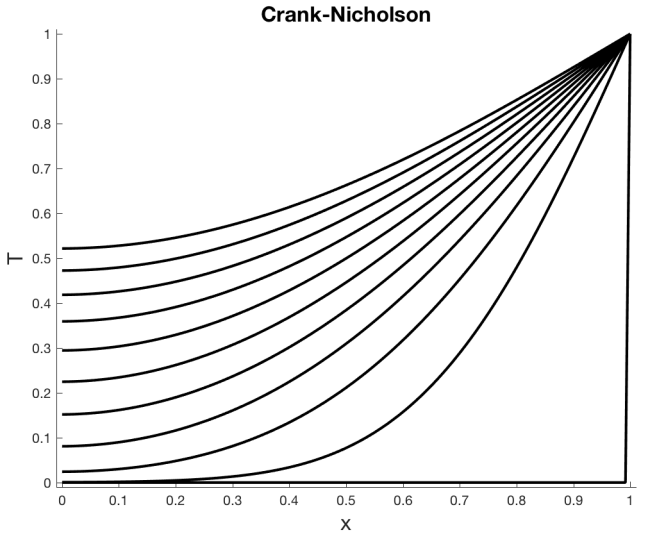
```
        % value at right boundary is one
        RHS(end) = RHS(end) - 1;
        result(i+1, :) = tridiag(n, mainDiagonal, lowerDiagonal, upperDiagonal, RHS
            ↪ );
    end
end
```

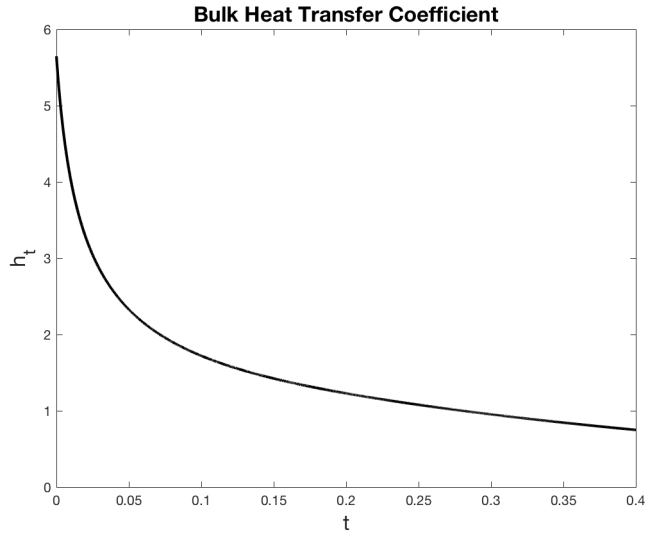
The following script now uses this Crank-Nicholson function to solve the heat diffusion equation given previously.

```
%% Problem 3
n = 121;
deltaX = 1/n;
x = linspace(0, 1-deltaX, n);
tFinal = 0.4;
alpha = 0.5;
deltaT = alpha*deltaX^2;
nTimeSteps = ceil(tFinal/deltaT);
deltaT = tFinal/nTimeSteps;

% initial conditions
T0Func = @(x) zeros(size(x));
T0 = T0Func(x);

% Boundary condtions
% zero flux at x = 0
Tr = 1;
RHSFunc = @(t, T) ([T(2:end),Tr]-2*T+[T(1),T(1:end-1)])/deltaX^2;

sol = crankNicholson(T0, nTimeSteps, deltaT, deltaX);

plotInterval = 0.04;
nPlots = ceil(tFinal/plotInterval);
timeStepInterval = floor(nTimeSteps/nPlots);
figure;
hold on
for n = 1:timeStepInterval:nTimeSteps
    plot([x,1], [sol(n,:),1], 'k', 'LineWidth', 2);
end
hold off
xlim([-0.01,1.01]);
ylim([-0.01,1.01]);
xlabel('x', 'FontSize', 16);
ylabel('T', 'FontSize', 16);
title('Crank-Nicholson', 'FontSize', 16);
saveas(gcf, 'Figures/03_05.png', 'png');

tStart = 0.01;
nInitial = ceil(tStart/deltaT);
ht = (ones(nTimeSteps+1 - nInitial+1,1) - sol(nInitial:end,end))/deltaX;
t = linspace(0,tFinal, nTimeSteps+1 - nInitial+1);
plot(t,ht, 'k', 'LineWidth', 2);
xlabel('t', 'FontSize', 16);
ylabel('h_t', 'FontSize', 16);
title('Bulk Heat Transfer Coefficient', 'FontSize', 16);
saveas(gcf, 'Figures/03_06.png', 'png');
```

The following two images are produced. The first image shows the temperature distribution over time. This solution makes sense because as more heat flows in from the right side and is unable to escape the insulated left side the solution approaches a steady state of constant

temperature. The bulk heat transfer plot shows that a lot of heat diffuses in to the slab at first, but then slows as the temperature gradient decreases.





4. (a) Is the scheme

$$U_i^{n+1} = U_i^n - \frac{C}{2}(U_{i+1}^n - U_{i-1}^n)$$

stable, conditionally stable or absolutely unstable? C is a constant.

In order to perform a Von Neumann Stability, each term needs to be replaced with an error approximation. Let ϵ_i^n be the error at x_i at time t_n. Approximating this error with white noise implies that $\epsilon_i^n = e^{\alpha t} e^{ik_m x_i}$. Now $\left|e^{\alpha \Delta t}\right|$ represent the growth of the error from one time step

to another. If $\left|e^{\alpha\Delta t}\right| \leq 1$, then the error decreases over time and the method is stable.

$$\epsilon_i^{n+1} = \epsilon_i^n - \frac{C}{2}(\epsilon_{i+1}^n - \epsilon_{i-1}^n) \tag{8}$$

$$e^{\alpha(t+\Delta t)}e^{ik_m x_i} = e^{\alpha t}e^{ik_m x_i} - \frac{C}{2}\Big(e^{\alpha t}e^{ik_m(x_i+\Delta x)} - e^{\alpha t}e^{ik_m(x_i-\Delta x)}\Big) \tag{9}$$

$$e^{\alpha\Delta t} = 1 - \frac{C}{2}\Big(e^{ik_m\Delta x} - e^{-ik_m\Delta x}\Big) \tag{10}$$

$$\left|e^{\alpha\Delta t}\right| = |1 - C(i\sin(k_m\Delta x))| \tag{11}$$

$$\left|e^{\alpha\Delta t}\right| = \sqrt{1 + C^2\sin^2(k_m\Delta x)} \tag{12}$$

$$\tag{13}$$

Note that this is always greater than one because $C^2\sin^2(k_m\Delta x) > 0$. This means that this method is absolutely unstable. That is there is no $C > 0$ where $\left|e^{\alpha\Delta t}\right| < 1$.

(b) Is the scheme

$$U_i^{n+1} = U_i^n - \frac{C}{2}\Big(U_{i+1}^{n+1} - U_{i-1}^{n+1}\Big)$$

stable, conditionally stable or absolutely unstable? C is a constant.
Again I will replace each term in the method with an error approximation.

$$\epsilon_i^{n+1} = \epsilon_i^n - \frac{C}{2}\Big(\epsilon_{i+1}^{n+1} - \epsilon_{i-1}^{n+1}\Big) \tag{14}$$

$$e^{\alpha(t+\Delta t)}e^{ik_m x_i} = e^{\alpha t}e^{ik_m x_i} - \frac{C}{2}\Big(e^{\alpha(t+\Delta t)}e^{ik_m(x_i+\Delta x)} - e^{\alpha(t+\Delta t)}e^{ik_m(x_i-\Delta x)}\Big) \tag{15}$$

$$e^{\alpha\Delta t} = 1 - e^{\alpha\Delta t}\frac{C}{2}\Big(e^{ik_m\Delta x} - e^{-ik_m\Delta x}\Big) \tag{16}$$

$$e^{\alpha\Delta t}\left(1 + \frac{C}{2}\Big(e^{ik_m\Delta x} - e^{-ik_m\Delta x}\Big)\right) = 1 \tag{17}$$

$$e^{\alpha\Delta t}(1 + Ci\sin(k_m\Delta x)) = 1 \tag{18}$$

$$e^{\alpha\Delta t} = \frac{1}{1 + Ci\sin(k_m\Delta x)} \tag{19}$$

$$\left|e^{\alpha\Delta t}\right| = \left|\frac{1}{1 + Ci\sin(k_m\Delta x)}\right| \tag{20}$$

$$\left|e^{\alpha\Delta t}\right| = \frac{1}{|1 + Ci\sin(k_m\Delta x)|} \tag{21}$$

$$\left|e^{\alpha\Delta t}\right| = \frac{1}{\sqrt{1 + C^2\sin^2(k_m\Delta x)}} < 1 \tag{22}$$

This method is unconditionally stable because for any C the change in error from one step to the next is scaled by a number less than 1.

(c) Which method would be called implicit?
The scheme in (b) would be called implicit as the value of U_i^{n+1} depends on the values U_{i+1}^{n+1} and U_{i-1}^{n+1}. The solution for a point at time t^{n+1} depends on the points next to it at the same time. This means that a system of equations must be solved in order to update the solution.