

Coursework 2

Hassan Miah

4254974

Contents

[Part A 3](#_Toc8595271)

[Part B 10](#_Toc8595272)

[References 32](#_Toc8595273)

[Appendix 33](#_Toc8595274)

[Figure 1: Part 1 of code 4](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584028)

[Figure 2: Part 2 of code 4](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584029)

[Figure 3: Part 3 of code 5](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584030)

[Figure 4: Part 4 of code 6](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584031)

[Figure 5: Loop 8](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584032)

[Figure 6: Part A results 9](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584033)

[Figure 7: Part 1 of code 11](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584034)

[Figure 8: Part 2 of code 12](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584035)

[Figure 9: Part 3 of code 12](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584036)

[Figure 10: Part 4 of code 13](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584037)

[Figure 11: Part 5 of code 17](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584038)

[Figure 12: Results of part B a 17](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584039)

[Figure 13: Part B, a 18](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584040)

[Figure 14: Part 1 of code 19](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584041)

[Figure 15: Part 2 of code 19](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584042)

[Figure 16: Part 3 of code 20](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584043)

[Figure 17: Part 4 of code 20](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584044)

[Figure 18: Part 5 of code 20](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584045)

[Figure 19: Part 6 of code 21](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584046)

[Figure 20: Part 7 of code 21](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584047)

[Figure 21: Part 8 of code 21](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584048)

[Figure 22: Part B,b results 22](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584049)

[Figure 23: Part 1 of code 24](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584050)

[Figure 24: Part 2 of code 24](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584051)

[Figure 25: Part 3 of code 25](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584052)

[Figure 26: Part 4 of code 25](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584053)

[Figure 27: Part 5 of code 25](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584054)

[Figure 28: Part 6 of code 26](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584055)

[Figure 29: Part 7 of code 26](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584056)

[Figure 30: Part 8 of code 26](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584057)

[Figure 31: Part B, c results 27](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584058)

[Figure 32: Part 1 of code 28](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584059)

[Figure 33: Part 2 of code 29](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584060)

[Figure 34: Part 3 of code 29](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584061)

[Figure 35: Part 4 of code 30](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584062)

[Figure 36: Part 5 of code 30](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584063)

[Figure 37: Part 6 of code 31](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584064)

[Figure 38: Part 7 of code 31](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584065)

[Figure 39: Part 8 of code 31](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584066)

[Figure 40: Part B, d results 32](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584067)

[Figure 41: Part A, a code 33](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584068)

[Figure 42: Part A, b code 34](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584069)

[Figure 43: Part B, a code 35](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584070)

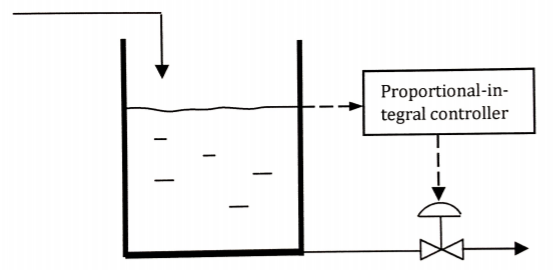
[Figure 44: Part B, b code 36](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584071)

[Figure 45: Part B, c code 37](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584072)

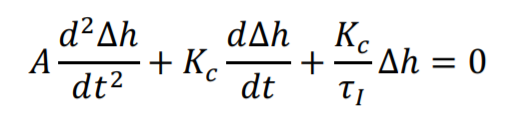
[Figure 46: Part B, d code 38](file:///C:\Users\Hassan\Documents\MATLAB\ACM%20cw2.docx#_Toc8584073)

## Part A

The liquid level of a tank is controlled at a desired value when the inlet flow rate undergoes step change. As shown in figure below, a feedback control system is used. This control system measures the liquid level and compares it with the desired steady-state value. If the level is higher than the desired value, it increases the effluent flow rate by opening the control valve, while it closes the valve when the level is lower than the desired value.



If the controller is a proportional-integral one, the following second-order ODE can be set up.



where A is the cross-sectional area of the tank (=2 m2), Kc is the proportional gain. τ1 is the integral time constant, t is time, and ∆ℎ is the deviation height of the liquid level in the tank, i.e., the difference between the actual height and the desired value. For a certain step change of the inlet flow rate, the following initial conditions can be written: at t = 0, , and ∆ℎ = 0.

(20%) Build your own solver using your own choice of method and solve the ODE if Kc and τ1 are set to 1 m2/min and 0.1 min, respectively. Plot ∆ℎ as a function of time and observe the dynamic behaviour of the liquid level of the tank.

(20%) Using the worksheet in part a, vary the value of Kc (for example from 1 to 5) to learn about the effect of this parameter on the oscillatory behaviour of the dynamic response of the liquid level due to a step change of the inlet flow rate. If the steady state value should be reached quickly, discuss whether or not a large value of Kc is better than a small one.

Euler method

MATLAB code:

Figure : Part 1 of code

clear all;

clc;

A = 2;

Taf1 = 0.1;

h = 0.01;

t\_final = 35;

N = t\_final/h;

Kc1 = 1;

t1(1) = 0;

v1(1) = 2;

x1(1) = 0;

Initial conditions are specified in figure 1 along with input variables found in the question.

for i = 1:N

t1(i+1) = t1(i)+h;

x1(i+1) = x1(i)+h\*(v1(i));

v1(i+1) = v1(i)+h\*((-Kc1\*v1(i)/A)-(Kc1/(taf1\*A))\*x1(i));

end

Figure : Part 2 of code

Figure 2 is a loop with a number of iterations providing values for the points on the graph.

v1(i+1)=

The differential values were calculated for each iteration.

x1(i+1)= x1(i)+h\*(v1(i))

This is the delta h value obtained for each iteration after the v1(i+1) is calculated; it is plugged in to the x1(i+1) obtaining values for each iteration. i = 1:N is the number of iterations used in the code that is calculated for. This is found by final time – the initial time and then divided by the number of grid points (gaps) on the graph to be plotted. The number of iterations is 35/0.01 = 3500 points in the graph therefore an accurate graph is produced.

Figure : Part 3 of code

Kc2 = 2;

t2(1) = 0;

v2(1) = 2;

x2(1) = 0;

for i = 1:N

t2(i+1) = t2(i)+h;

x2(i+1) = x2(i)+h\*(v2(i));

v2(i+1) = v2(i)+h\*((-Kc2\*v2(i)/A)-(Kc2/(taf1\*A))\*x2(i));

end

Kc3 = 3;

t3(1) = 0;

v3(1) = 2;

x3(1) = 0;

for i = 1:N

t3(i+1) = t3(i)+h;

x3(i+1) = x3(i)+h\*(v3(i));

v3(i+1) = v3(i)+h\*((-Kc3\*v3(i)/A)-(Kc3/(taf1\*A))\*x3(i));

end

Kc4 = 4;

t4(1) = 0;

v4(1) = 2;

x4(1) = 0;

for i = 1:N

t4(i+1) = t4(i)+h;

x4(i+1) = x4(i)+h\*(v4(i));

v4(i+1) = v4(i)+h\*((-Kc4\*v4(i)/A)-(Kc4/(taf1\*A))\*x4(i));

end

Kc5 = 5;

t5(1) = 0;

v5(1) = 2;

x5(1) = 0;

for i = 1:N

t5(i+1) = t5(i)+h;

x5(i+1) = x5(i)+h\*(v5(i));

v5(i+1) = v5(i)+h\*((-Kc5\*v5(i)/A)-(Kc5/(taf1\*A))\*x5(i));

end

The code in figure 2 is repeated 5 times with Kc varied between 1 and 5.

Figure : Part 4 of code

subplot(3,2,1)

plot(t1,x1,'r')

title('Kc = 1')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,2)

plot( t2 ,x2, 'g')

title('Kc = 2')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,3)

plot( t3, x3,'b')

title('Kc = 3')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,4)

plot(t4, x4,'y')

title('Kc = 4')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,5)

plot(t5, x5,'k')

title('Kc = 5')

xlabel('Time(s)')

ylabel('delta h')

grid on

Figure 4 contains the code to plot a subplot of 5 graphs which shows the graphs of Kc = 1, 2, 3, 4 and 5.

Initial

Set parameters:

A = 2, taf1 = 0.1, h = 0.1, t\_final = 35, N = t\_fianl/h, Kc = 1.

Initial conditions:

t1(1) = 0, v1(1) = 2 and x1(1) = 0.

Equation: t1(i+1)

Example:

1st iteration t1(2) = 0.01

2nd iteration t1(3) = 0.02

Loop ends at i = 3500

Equation: v1(i+1)

Example:

1st iteration v1(2) = 1.99

2nd iteration v1(3) = 1.9791

Loop ends at i = 3500

Equation: x1(i+1)

Example:

1st iteration x1(2) = 0.02

2nd iteration x1(3) = 0.0399

Loop ends at i = 3500

Plot

Figure : Loop



Figure : Part A results

Figure 6 displays what happens when Kc is varied from 1 to 5; as Kc increases, the stability increases this means the delta h stops changing in a shorter time. Time for stability decreases from 25 seconds to 5 seconds. The maximum delta h (amplitude of the wave) also decreases from 0.75 to 0.3.

The deviation from liquid level of tank from its set point is represented by the graphs in figure 6.

## Part B

In many engineering applications, advection (or convection) and diffusion are the dominant physical transport mechanisms over much of the domain of interest. Consider the case of a plume of contaminant being transported in flowing river. The well-known governing equation for such transport is the advection-diffusion equation below



where c is the concentration of the plume, D is the diffusivity (or diffusion coefficient), v is the velocity the concentration is moving.

• Solve the problems below by generating your own finite difference method and without resorting to any built-in MATLAB solvers.

Initial value problems:

You could imagine this problem as when a plume of contaminant is released at an initial time on a moving bed of fluid.

1. Consider a one-dimensional problem where diffusion effects are omitted. The equation above reduces to



The initial distribution (t = 0) of concentration c0 is a Gaussian i.e.

. On the domain 0 < x < 2, with constant velocity v = 1, the initial profile of c, as well as c at t = 0.5, and 1.

Figure : Part 1 of code

clear all;

clc;

close all;

%numerical grid

xlength = 2; %grid length

n = 500; %number of grid points

h = xlength/(n-1); %gap between grid points

x = zeros(1,n);

% set numerical & physical parameters

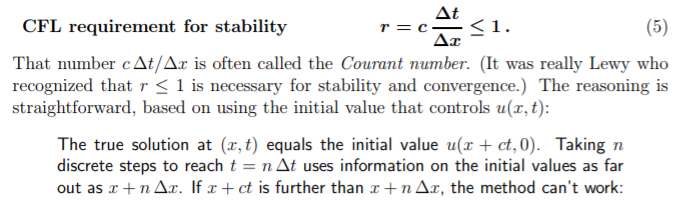
cfl = 0.9;

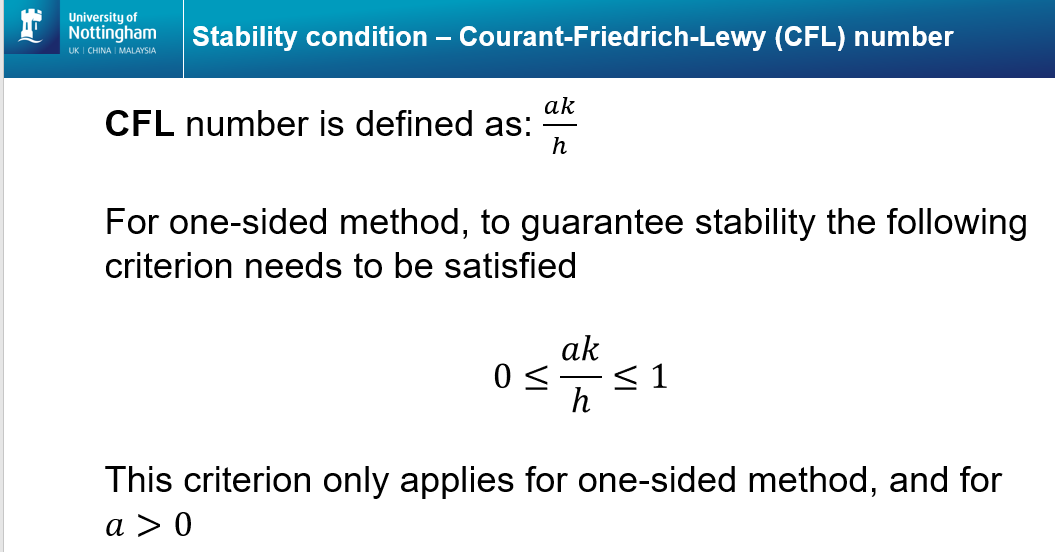
a = 1d0;

dt = h \* cfl / a;

tout = 1;

Figure 7 contains the input variables.





for time = [0 0.5 1];

x = zeros (1,n);

fn = zeros (1,n);

fnlw = zeros (1,n);

f = zeros (1,n);

flw = zeros (1,n);

x(1) = 0D0;

Figure : Part 2 of code

for i=2:n; % assigning values to array x(i)

x(i)=x(i-1)+h;

end

for i = 1:n;

f(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2);

flw(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2);

end

Figure : Part 3 of code

nt = time/dt;

for k = 1:nt

for i = 2:n %first order upwind

flux = a \* (f(i)-f(i-1));

fn(i) = f(i)-(dt/h)\*flux;

end

fn(1) = fn(n);

f = fn;

for i = 2:n-1 %Lax-Wendroff

l0 = (dt/(2\*h))\* a \* (flw(i+1)-flw(i-1));

h0 = (dt^2/(2\*h^2))\* a^2 \* (flw(i+1)-(2\*flw(i))+flw(i-1));

fnlw(i) = flw(i) - l0 + h0;

end

fnlw(1) = fn(n);

fnlw(n) = fn(1);

flw = fnlw;

end

hold on

Figure : Part 4 of code

Higher order methods used to solve the question in part B are:

* One-sided (first order upwind)
* Lax-Wendroff

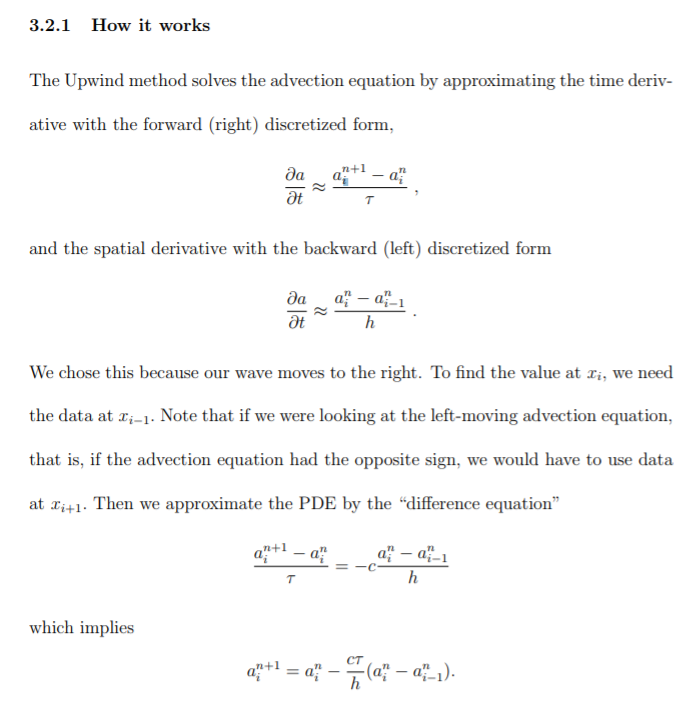
First order upwind:

How it works

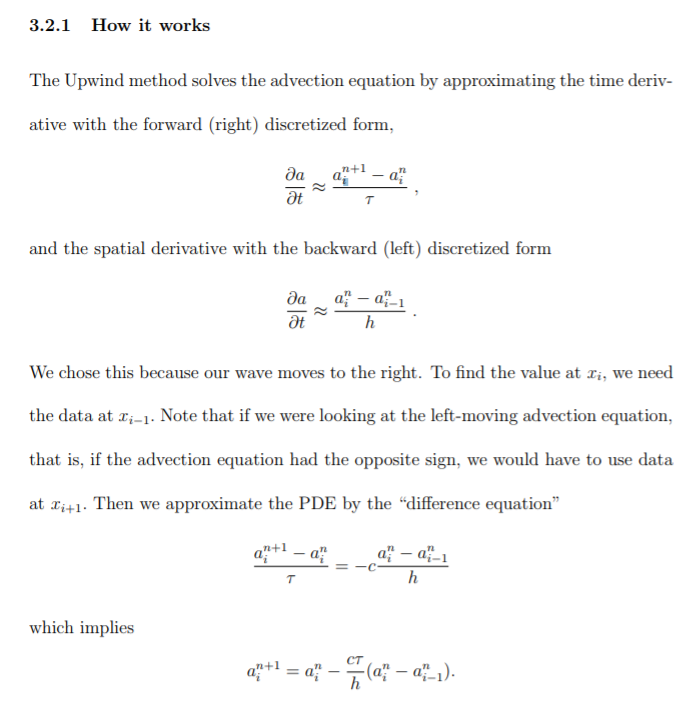
The Upwind method solves the advection equation by approximating the time derivative with the forward (right) discretized form,



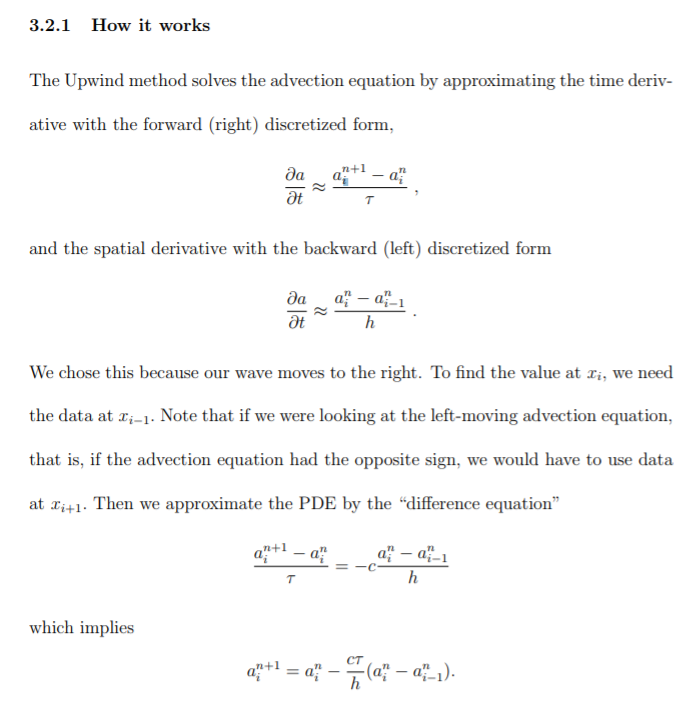
and the spatial derivative with the backward (left) discretized form



We chose this because our wave moves to the right. To find the value at xi , we need the data at xi−1. Note that if we were looking at the left-moving advection equation, that is, if the advection equation had the opposite sign, we would have to use data at xi+1. Then we approximate the PDE by the “difference equation”



Which implies



As with the FTCS method, we cannot compute the boundary points using this equation so we must again use boundary conditions. 2 For a graphical version, see Figure 6.



Figure 6: Computational Molecule for Upwind Method

Again, each point represents a value of the function a. The black points are the ones used in the equation and the grey points are just other points found on the grid.

So as long as this condition is true, the Upwind method is stable. This is actually known as the Courant condition, or the Courant-Friedrichs-Lewy (CFL) condition and the fraction cτ h is called a Courant factor.

f(i) = =

Flux =

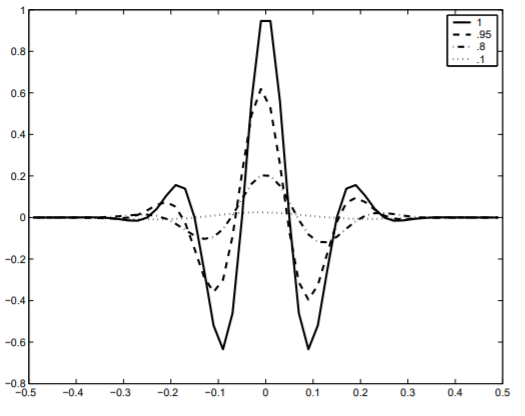
fn(i) =

dt = h \* cfl / a

fn(i) = f(i)-(dt/h)\*flux;

Solutions

The advantage of the Upwind method is its simplicity. However, the severe dissipation and first-order error in time and space make this method less desirable. In addition, as we stated before, Upwind only works for waves moving in one direction (since it only takes into account the forward time, backward space differences) and cannot be used for second-order wave equations. Therefore, we will consider other methods.



Lax – Wendroff:

How it works

For this method, we will approximate our function by the Taylor expansions (through the second-order term) where the time derivatives are replaced by the discretized centred differences.

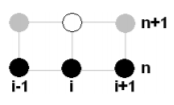
l0 =

h0 =

flw(i) =

fnlw(i) = flw(i) - l0 + h0 =

4. Lax-Wendroff: (14) has extra accuracy, conditionally stable also for c < 0





subplot(2,1,1)

plot(x,f)

xlabel('Time (min)')

ylabel('Concentration')

legend('upwind')

shg

pause(dt)

hold on

subplot(2,1,2)

plot(x,flw)

xlabel('Time (min)')

ylabel('Concentration')

legend('LWendroff')

shg

pause(dt)

end



Figure : Results of part B, a when cfl = 0.5

Figure : Results of part B, a when cfl = 0.9

Figure 12 shows the concentration profile of a one-dimensional problem where a plume of contaminant is released at t = 0, 0.5 and 1 in a flowing river, only advection occurs and no diffusion. The initial distribution is expressed by the equation at t = 0, this is represented by the blue graph in both Lax-Wendroff and First Order Upwind. The red graph represents t = 0.5 it can be seen on the first order upwind graph is shifted down to 0.7355 and on the Lax-Wendroff it remains at 0.7499 concentration. The yellow graph is the profile at t = 1 and the maximum concentration further drops to 0.7217 on the First Order Upwind and remains constant in the Lax-Wendroff graph. we see that this method does not dampen our solution like the previous methods (that is, there is less dissipation), but it does shift the solution slightly with decreasing values of α (there is dispersion) (Lynch, 2004). The cfl number = 0.9 in this code, cfl should be between 0 and 1. The lower the cfl number the more accurate the graph is as this increases the number of time intervals nt; in this code nt = 277 if the cfl was lowered to 0.5 the nt = 499. A higher nt increases the number of points on the graph hence it is more accurate. However, lowering the cfl number increases the dampening affect on the First Order Upwind graph for t = 0.5 and 1 as shown in figure 11. Lax-Wendroff method is a second-order difference method in both time and space this considers second order difference unlike First Order Upwind.

Input

Set parameters for xlength =2, n = 500, h = xlength/n-1, x = zeros (1,n), cfl = 0.9, a = 1d0, dt = h\*cfl/a.

i = 2:n-1

Lax-Wendroff

i = 2:n

First order upwind

Time = [0, 0.5, 1]

Subplot

K = 1:nt

i = 2:n

i = 1:n

Figure : Part B, a

1. Now consider the case where the migration of the plume is subject only to diffusion process with D = 1. Still with the same initial condition, plot the concentration profiles at t = 0.5, and 1.

clc % Clear the command window

close all % Close all previously opened figure windows

clear all % Clear all previously generated variables

N = 41; % Number of nodes

nStep = 500; % Number of time levels

L = 2.0; % Domain length

dx = L/(N-1); % Node spacing

dt = 0.0005; % Time step

D = 1; % Diffusion coefficient

A = 1; % Amplitude of the initial sine wave

k = 1; % Frequency of the initial sine wave

U = 0; % Advection speed

f = zeros(N,1); % Calculated solution at the new time level (level n+1)

fOld = zeros(N,1); % Calculated solution at the previous time level (level n)

exact = zeros(N,1); % Exact solution

time = 0.0;

Figure : Part 1 of code

Figure 13 contains the input variables. The diffusion coefficient is 1 as this process is subject to diffusion only and no advection occurs therefore advection is 0 (U=0).

Figure : Part 2 of code

% Use the given initial condition

for j = 1:N % Node counter

x = dx\*(j-1); % x coordinate of node i

fOld(j) = 0.75\*exp(-((x-0.5)/0.1).^2);

end

fprintf('Watch the progress of the solution in the figure window.\n\n')

fprintf('Press Ctrl-C in the command line window to terminate the code.\n\n')

Figure 14 counts the number of iterations for time and displays in command window.

Figure : Part 3 of code

for n = 1:nStep % Time loop. n is the time level counter

G=n\*dt

if G<=1

time = time+dt

end

Figure 15 contains the number of iterations

% Calculate the solution of this time level, i.e. f\_n+1, for inner points

for j = 2:N-1

f(j) = fOld(j) - U\*dt/(2\*dx) \* (fOld(j+1) - fOld(j-1)) + ...

D\*dt/dx^2 \* (fOld(j+1) - 2\*fOld(j) + fOld(j-1));

end

Figure : Part 4 of code

Figure 16

Figure : Part 5 of code

% Calculate the solution of the last point (Right boundary)

f(N) = fOld(N) - U\*dt/(2\*dx) \* (fOld(2) - fOld(N-1)) + ...

D\*dt/dx^2 \* (fOld(2) - 2\*fOld(N) + fOld(N-1));

f(1) = f(N); % Due to the periodic BC

hold off;

plot(f, 'linewidth',2); % Plot the calculated solution

grid on;

axis([1 N 0, 1.0]);

xlabel('Node number');

ylabel('f');

title('1D Diffusion Solution with Periodic BCs')

Figure : Part 6 of code

Figure : Part 7 of code

hold on;

plot(exact,'r--','linewidth',2); % Plot exact solution of the previous time step

legend('Numerical','Exact');

pause(0.01);

% Calculate L2 error

Error = dx \* sqrt(sum((f - exact).^2))

% Get ready for the new time level before the loop ends

fOld = f;

end % End of the time loop

fprintf('Done. The code is terminated successfuly.\n\n')

Figure : Part 8 of code



Figure : Part B, b results

Figure 22 shows the migration of the plume in the river subject only to diffusion. The concentration varies across a distance which starts at 0 m displacement up to 2 m. The blue line graph represents the concentration profile at t = 0.5 s and its maximum concentration is 0.02727 at distance 0.965 m. The red line graph represents the concentration profile at t = 1 s and its maximum concentration is 0.007923 at distance 1 m.

1. Finally, consider the case where the migration of the plume is subject to both advection, and diffusion. With all the parameters above maintained the same, plot the concentration profiles at t = 0.5, and 1.

clc % Clear the command window

close all % Close all previously opened figure windows

clear all % Clear all previously generated variables

N = 41; % Number of nodes

nStep = 5000; % Number of time levels

L = 2.0; % Domain length

dx = L/(N-1); % Node spacing

dt = 0.005; % Time step

D = 0.04; % Diffusion coefficient

A = 1; % Amplitude of the initial sine wave

k = 1; % Frequency of the initial sine wave

U = 1; % Advection speed

f = zeros(N,1); % Calculated solution at the new time level (level n+1)

fOld = zeros(N,1); % Calculated solution at the previous time level (level n)

exact = zeros(N,1); % Exact solution

time = 0.0;

Figure : Part 1 of code

Figure : Part 2 of code

% Use the given initial condition

for j = 1:N % Node counter

x = dx\*(j-1); % x coordinate of node i

fOld(j) = 0.75\*exp(-((x-0.5)/0.1).^2);

end

fprintf('Watch the progress of the solution in the figure window.\n\n')

fprintf('Press Ctrl-C in the command line window to terminate the code.\n\n')

for n = 1:nStep % Time loop. n is the time level counter

G=n\*dt

if G<=1

time = time+dt

end

% Calculate the solution of this time level, i.e. f\_n+1, for inner points

for j = 2:N-1

f(j) = fOld(j) - U\*dt/(2\*dx) \* (fOld(j+1) - fOld(j-1)) + ...

D\*dt/dx^2 \* (fOld(j+1) - 2\*fOld(j) + fOld(j-1));

end

Figure : Part 3 of code

% Calculate the solution of the last point (Right boundary)

f(N) = fOld(N) - U\*dt/(2\*dx) \* (fOld(2) - fOld(N-1)) + ...

D\*dt/dx^2 \* (fOld(2) - 2\*fOld(N) + fOld(N-1));

f(1) = f(N); % Due to the periodic BC

Figure : Part 4 of code

hold off;

plot(f, 'linewidth',2); % Plot the calculated solution

grid on;

axis([1 N 0, 1.0]);

xlabel('Node number');

ylabel('f');

title('1D Advection-Diffusion Solution with Periodic BCs')

title(sprintf('time= %1.2f',time),'fontsize',16)

Figure : Part 5 of code

hold on;

plot(exact,'r--','linewidth',2); % Plot exact solution of the previous time step

legend('Numerical','Exact');

pause(0.00001);

Figure : Part 6 of code

% Calculate L2 error

Error = dx \* sqrt(sum((f - exact).^2))

Figure : Part 7 of code

% Get ready for the new time level before the loop ends

fOld = f;

end % End of the time loop

fprintf('Done. The code is terminated successfuly.\n\n')

Figure : Part 8 of code



Figure : Part B, c results

Figure 31 shows the migration of the plume in the river subject to diffusion and advection. The concentration varies across a distance which starts at 0 m displacement up to 2 m. The blue line graph represents the concentration profile at t = 0.5 s and its maximum concentration is 0.03169 at distance 1.155 m. The red line graph represents the concentration profile at t = 1 s and its maximum concentration is 0.008295 at distance 1.2 m.

Boundary value problems:

(20%) Consider the case where the migration of contaminant is subject to both advection, and diffusion. This time, instead of a given initial plume profile, the concentration (of contaminant) at the inlet boundary is maintained at c(x=0) = 1. Solve the contaminant transport problem and plot the concentration profile along x at t = 0, 0.25, 0.5, 1, 2, 3.

nStep = 500; % Number of time levels

L = 2.0; % Domain length

dx = L/(N-1); % Node spacing

dt = 0.0005; % Time step

D = 1; % Diffusion coefficient

A = 1; % Amplitude of the initial sine wave

k = 1; % Frequency of the initial sine wave

U = 0; % Advection speed

f = zeros(N,1); % Calculated solution at the new time level (level n+1)

fOld = zeros(N,1); % Calculated solution at the previous time level (level n)

exact = zeros(N,1); % Exact solution

time = 0.0;

Figure : Part 1 of code

% Use the given initial condition

for j = 1:N % Node counter

x = dx\*(j-1); % x coordinate of node i

fOld(j) = 1;

end

fprintf('Watch the progress of the solution in the figure window.\n\n')

fprintf('Press Ctrl-C in the command line window to terminate the code.\n\n')

for n = 1:nStep % Time loop. n is the time level counter

G=n\*dt

if G<=3

time = time+dt

end

Figure : Part 2 of code

Figure : Part 3 of code

% Calculate the solution of this time level, i.e. f\_n+1, for inner points

for j = 2:N-1

f(j) = fOld(j) - U\*dt/(2\*dx) \* (fOld(j+1) - fOld(j-1)) + ...

D\*dt/dx^2 \* (fOld(j+1) - 2\*fOld(j) + fOld(j-1));

end

% Calculate the solution of the last point (Right boundary)

f(N) = fOld(N) - U\*dt/(2\*dx) \* (fOld(2) - fOld(N-1)) + ...

D\*dt/dx^2 \* (fOld(2) - 2\*fOld(N) + fOld(N-1));

f(1) = 1;

f(N)=0; % Due to the periodic BC

Figure : Part 4 of code

hold off;

plot(f, 'linewidth',2); % Plot the calculated solution

grid on;

axis([1 N 0, 1.0]);

xlabel('Node number');

ylabel('f');

title('1D Advection-Diffusion Solution with Periodic BCs')

Figure : Part 5 of code

hold on;

plot(exact,'r--','linewidth',2); % Plot exact solution of the previous time step

legend('Numerical','Exact');

pause(0.01);

Figure : Part 6 of code

% Calculate L2 error

Error = dx \* sqrt(sum((f - exact).^2))

Figure : Part 7 of code

% Get ready for the new time level before the loop ends

fOld = f;

end % End of the time loop

fprintf('Done. The code is terminated successfuly.\n\n')

Figure : Part 8 of code



Figure : Part B, d results

Figure 40 shows the migration of the plume in the river subject to diffusion and advection however this time the concentration at the source is at a constant concentration = 1. There are 6 graphs with different times for t = 0, 0.25, 0.5, 1, 2 and 3. The concentration varies across a distance which starts at 0 m displacement up to 2 m. The graph t = 0 concentration profile is c(x=0) = 1 as stated in the question; the concentration does not change and remains constant at 1 throughout the distance from 0 to 2 m. The other 5 graphs are t = 0.25, 0.5, 1, 2 and 3 where the concentration is 0 after 2 m.

## References

<https://ocw.mit.edu/courses/mathematics/18-086-mathematical-methods-for-engineers-ii-spring-2006/readings/am52.pdf>

## Appendix

clear all;

clc;

A = 2; % cross-sectional area (m2)

taf1 = 0.1; % ?1 (min)

h = 0.01; %gap between grid points

t\_final = 35; % final time

N = t\_final/h; % Number of grid points

Kc1 = 1; % (m2/min), proportional gain

% initial condtions:

t1(1) = 0; % time

v1(1) = 2; % velocity

x1(1) = 0; % delta h

for i = 1:N % number of iterations

t1(i+1) = t1(i)+h; % time at an interval

x1(i+1) = x1(i)+h\*(v1(i)); % delta h at an interval

v1(i+1) = v1(i)+h\*((-Kc1\*v1(i)/A)-(Kc1/(taf1\*A))\*x1(i)); %

end

Figure : Part A, a code

Figure : Part A, b code

Kc2 = 2;

t2(1) = 0;

v2(1) = 2;

x2(1) = 0;

for i = 1:N

t2(i+1) = t2(i)+h;

x2(i+1) = x2(i)+h\*(v2(i));

v2(i+1) = v2(i)+h\*((-Kc2\*v2(i)/A)-(Kc2/(taf1\*A))\*x2(i));

end

Kc3 = 3;

t3(1) = 0;

v3(1) = 2;

x3(1) = 0;

for i = 1:N

t3(i+1) = t3(i)+h;

x3(i+1) = x3(i)+h\*(v3(i));

v3(i+1) = v3(i)+h\*((-Kc3\*v3(i)/A)-(Kc3/(taf1\*A))\*x3(i));

end

Kc4 = 4;

t4(1) = 0;

v4(1) = 2;

x4(1) = 0;

for i = 1:N

t4(i+1) = t4(i)+h;

x4(i+1) = x4(i)+h\*(v4(i));

v4(i+1) = v4(i)+h\*((-Kc4\*v4(i)/A)-(Kc4/(taf1\*A))\*x4(i));

end

Kc5 = 5;

t5(1) = 0;

v5(1) = 2;

x5(1) = 0;

for i = 1:N

t5(i+1) = t5(i)+h;

x5(i+1) = x5(i)+h\*(v5(i));

v5(i+1) = v5(i)+h\*((-Kc5\*v5(i)/A)-(Kc5/(taf1\*A))\*x5(i));

end

subplot(3,2,1)

plot(t1,x1,'r')

title('Kc = 1')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,2)

plot( t2 ,x2, 'g')

title('Kc = 2')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,3)

plot( t3, x3,'b')

title('Kc = 3')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,4)

plot(t4, x4,'y')

title('Kc = 4')

xlabel('Time(s)')

ylabel('delta h')

grid on

subplot(3,2,5)

plot(t5, x5,'k')

title('Kc = 5')

xlabel('Time(s)')

ylabel('delta h')

grid on

Figure : Part B, a code

clear all;

clc;

close all;

% numerical grid

xlength = 2; % time length on graph, (x-axis)

n = 500; % number of grid points

h = xlength/(n-1); % gap between grid points

x = zeros(1,n);

% set numerical & physical parameters

cfl = 0.5; % Courant–Friedrichs–Lewy

a = 1d0;

dt = h \* cfl / a; % change in time

for time = [0 0.5 1]; % plume released at these time intervals

x = zeros (1,n);

fn = zeros (1,n);

fnlw = zeros (1,n);

f = zeros (1,n);

flw = zeros (1,n);

x(1) = 0D0;

% assigning values to array x(i)

for i=2:n; % 499 iterations

x(i)=x(i-1)+h;

end

for i = 1:n;

f(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2); % distribution of conc. with time

flw(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2);

end

nt = time/dt; % number of time intervals

for k = 1:nt

for i = 2:n %first order upwind

flux = a \* (f(i)-f(i-1));

fn(i) = f(i)-(dt/h)\*flux;

end

fn(1) = fn(n);

f = fn;

for i = 2:n-1 %Lax-Wendroff

l0 = (dt/(2\*h))\* a \* (flw(i+1)-flw(i-1));

h0 = (dt^2/(2\*h^2))\* a^2 \* (flw(i+1)-(2\*flw(i))+flw(i-1));

fnlw(i) = flw(i) - l0 + h0;

end

fnlw(1) = fn(n);

fnlw(n) = fn(1);

flw = fnlw;

end

hold on

% graph plot

subplot(2,1,1)

plot(x,f)

xlabel('Time (min)')

ylabel('Concentration')

legend('upwind')

shg

pause(dt)

hold on

subplot(2,1,2)

plot(x,flw)

xlabel('Time (min)')

ylabel('Concentration')

legend('LWendroff')

shg

pause(dt)

end

Figure : Part B, b code

clear all;

clc;

close all;

%this code solves, and simulates a scalar conservation law equation of

% df(x,t)/dt + a df(x,t)/dx = 0

hold on

%numerical grid

xlength=2; %grid length, upper limit of domain set to 2

n=1000; %number of grid points

h=xlength/(n-1); %gap between grid points

% set numerical and physical parameters

D=1;

U=0;

dt = 0.000001;

x = zeros(1,n);

f = zeros(1,n);

fn = zeros(1,n);

freal= zeros(1,n);

x(1) = 0D0;

% assigning values to array x(i)

for i=2:n

x(i)=x(i-1)+h;

end

% initialising function

for i = 1:n

f(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2);

end

for time= [0.5:0.5:1] %creates time matrix values with increments of 0.5

nt = 0.5/dt; %find out %number of iterations (nt)

for k = 1:nt

for i = 2:n-1

%Determining difference equation for diffusion-only situation

% flux= D \* (f(i+1)-(2\*f(i))+f(i-1));

fn(i) = f(i)-((U\*(dt/(2\*h)))\*(f(i+1)-f(i-1)))+((D\*(dt/(h^2)))\*(f(i+1)-(2\*f(i))+f(i-1)));

end

%boundary condition

fn(1) = fn(n);

fn(n) = fn(1);

f = fn;

end

warning off

plot(x,f) %Plots Distance vs Concentration for diffusion-only situation

grid on

xlabel('Distance (m)');

ylabel('Concentration (c)')

legend('t = 0.5','t = 1')

end

hold off;

Figure : Part B, c code

clear all;

clc;

close all;

%this code solves, and simulates a scalar conservation law equation of

% df(x,t)/dt + a df(x,t)/dx = 0

hold on

%numerical grid

xlength=2; %grid length, upper limit of domain set to 2

n=1000; %number of grid points

h=xlength/(n-1); %gap between grid points

% set numerical and physical parameters

D=1;

U=1;

dt = 0.000001;

x = zeros(1,n);

f = zeros(1,n);

fn = zeros(1,n);

freal= zeros(1,n);

x(1) = 0D0;

% assigning values to array x(i)

for i=2:n

x(i)=x(i-1)+h;

end

% initialising function

for i = 1:n

f(i) = 0.75\*exp(-((x(i)-0.5)/0.1).^2);

end

for time= [0.5:0.5:1] % creates time matrix values with increments of 0.5

nt = 0.5/dt; % find out %number of iterations (nt)

for k = 1:nt

for i = 2:n-1

% Determining difference equation for diffusion-only situation

% flux= D \* (f(i+1)-(2\*f(i))+f(i-1));

fn(i) = f(i)-((U\*(dt/(2\*h)))\*(f(i+1)-f(i-1)))+((D\*(dt/(h^2)))\*(f(i+1)-(2\*f(i))+f(i-1)));

end

% boundary condition

fn(1) = fn(n);

fn(n) = fn(1);

f = fn;

end

warning off

% Plots Distance vs Concentration for diffusion-only situation

plot(x,f)

grid on

xlabel('Distance (m)');

ylabel('Concentration (c)')

legend('t = 0.5','t = 1')

% shg

% pause(0.1)

end

hold off;

clear all;

clc;

close all;

%this code solves, and simulates a scalar conservation law equation of

% df(x,t)/dt + a df(x,t)/dx = 0

hold on

%numerical grid

xlength=2; %grid length, upper limit of domain set to 2

n=1000; %number of grid points

h=xlength/(n-1); %gap between grid points

% set numerical and physical parameters

D=1;

U=1;

dt = 0.000001;

x = zeros(1,n);

f = zeros(1,n);

fn = zeros(1,n);

freal= zeros(1,n);

x(1) = 0D0;

% assigning values to array x(i)

for i=2:n

x(i)=x(i-1)+h;

end

% initialising function

for i = 1:n

f(i) = 1;

end

for time= [0 0.25 0.5 1 2 3] %creates time matrix values with increments of 0.5

nt = time/dt; %find out %number of iterations (nt)

for k = 1:nt

for i = 2:n-1

%Determining difference equation for diffusion-only situation

% flux= D \* (f(i+1)-(2\*f(i))+f(i-1));

fn(i) = f(i)-((U\*(dt/(2\*h)))\*(f(i+1)-f(i-1)))+((D\*(dt/(h^2)))\*(f(i+1)-(2\*f(i))+f(i-1)));

end

%boundary condition

fn(1) = 1;

fn(n) = 0;

f = fn;

end

warning off

plot(x,f) %Plots Distance vs Concentration for diffusion-only situation

grid on

xlabel('Distance (m)');

ylabel('Concentration (c)')

legend('t = 0','t = 0.25','t = 0.5','t = 1','t = 2','t = 3')

% shg

% pause(0.1)

end

hold off;

Figure : Part B, d code