Numerical Methods for Complex Systems
Auralius Manurung
ME - Universitas Pertamina

Table of Contents

1. Case 1: 1-D Non-Homogenous Linear Advection[5]

Given a hyperbolic PDE as follows.

$$u_t + cu_x = f(x,t) \tag{1}$$

with the following initial condition:

$$u(x,0) = u_0(x) \tag{2}$$
$$c = \text{constant of real number}$$

We want to to find its solution numerically. When $f(x,t) = 0$, the analytical solution is given by: $u(x,t) = u_0(x - ct)$. Basically, u_0 propagates along x with a constant speed c. This kind of PDE is widely used for real time modeling and control application. Therefore, it is important to investigate its non-homogenous version to further allow an implementation of a PDE system. A PDE system is composed of more than one interacting PDEs. For example, in a heat exchanger, there are at least two PDEs: one for the hot system and another one for the cold system. The two PDEs interact by exchanging heat between them.

1.1. Forward in Time Centered in Space (FTCS)

This method is a finite difference method with central difference for the distance to increase the approximation. To approximate u_t, we use the following approximation.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{3}$$

While to approach u_x, we use the following approximation.

$$u_x \approx \frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x} \tag{4}$$

Substituting (3) and (4) to (1):

$$\begin{aligned}
u_t &\approx -cu_x + f(x,t) \\
\frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} &\approx -c\frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x} + f(x,t) \\
u(x,t+\Delta t) - u(x,t) &\approx -\frac{c\Delta t}{2\Delta x}\Big[u(x+\Delta x,t) - u(x-\Delta x,t)\Big] + f(x,t)\Delta t \\
u(x,t+\Delta t) &\approx u(x,t) - \frac{c\Delta t}{2\Delta x}\Big[u(x+\Delta x,t) - u(x-\Delta x,t)\Big] + f(x,t)\Delta t
\end{aligned} \tag{5}$$

FTCS method is unconditionally unstable. It should never be used. Nevertheless, FTCS is the fundamental building block for another methods, such as the Lax method.

1.2. Upwind Method ($c > 0$)

By using finite difference method, we can approximate both u_t and u_x. To approximate u_t, we use the following approximation.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{6}$$

While to approach u_x, we use the following approximation.

$$u_x \approx \frac{u(x,t) - u(x-\Delta x,t)}{\Delta x} \tag{7}$$

Substituting (6) and (7) to (1):

$$\begin{aligned} u_t &\approx -cu_x + f(x,t) \\ \frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} &\approx -c\frac{u(x,t)-u(x-\Delta x,t)}{\Delta x} + f(x,t) \\ u(x,t+\Delta t)+u(x,t) &\approx -\frac{c\Delta t}{\Delta x}\Big[u(x,t)-u(x-\Delta x,t)\Big] + f(x,t) \\ u(x,t+\Delta t) &\approx u(x,t)-\frac{c\Delta t}{\Delta x}\Big[u(x,t)-u(x-\Delta x,t)\Big] + f(x,t)\Delta t \end{aligned} \tag{8}$$

Consider $\frac{c\Delta t}{\Delta x} = r$, $U_{j,k} = u(x_j, t_k)$, $F_{j,k} = f(x_j, t_k)$, $x_j = j\Delta x$ and $t_k = k\Delta t$, we can have the following shorter terms:

$$\boxed{U_{j,k+1} = U_{j,k} - r(U_{j,k} - U_{j-1,k}) + F_{j,k}\Delta t} \tag{9}$$

MATLAB implementation of an upwind method (Eq. 9) is as follows.

```
function next_u_array = upwind(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

for i = 2:N
    next_u_array(i) = u_array(i)-c*dt/dx*(u_array(i)-u_array(i-1))+f_array(i)*dt;;
end

end
```

To increase its efficiency, the MATLAB code above can be vectorized as follows.

```
function next_u_array = upwind(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

next_u_array(2:N) = u_array(2:N) - (c*dt/dx).* ...
                    (u_array(2:N)- u_array(1:N-1)) + f_array(2:N).*dt;

end
```

The following lines of code is used to test the upwind function. Here we want to to solve the following PDE.

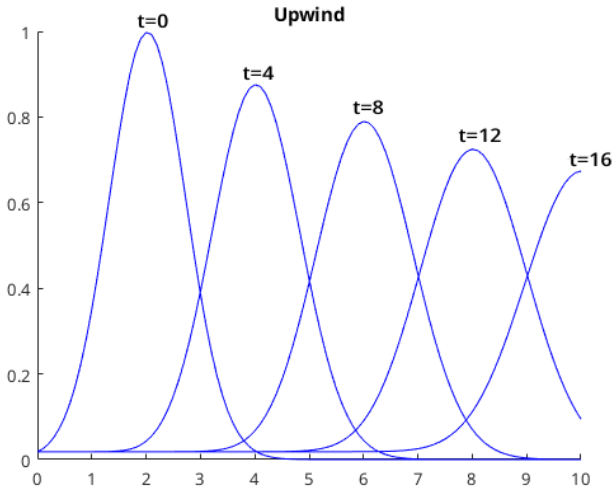
$$\begin{gathered} u_t + 0.5u_x = 0 \\ u(x,0) = e^{-(x-2)^2} \\ 0 \leqslant x \leqslant 10 \end{gathered} \tag{10}$$

The length is divided 100 segments ($dx = 0.1$) and the time is divided into 400 segments ($dt = 0.05$).

```
L = 10;
dx = 0.1;
x = 0:dx:L;

dt = 0.05;
T = 20;
t = 0:dt:T;
```

```
8
9  c = 0.5; % Upwind
10
11 u = zeros(length(x),1);
12 f = zeros(length(x),1);
13
14 for k = 1:length(x)
15     u(k) = exp(-(x(k)-2).^2);
16 end
17
18 figure;
19 h = plot(0,0);
20 title('Upwind')
21 ylim([0 1]);
22
23 for k = 1:length(t)
24     u = upwind(u,f, c, dt, dx);
25     set(h,'XData',x,'Ydata',u);
26     drawnow;
27 end
```

The analytical solution of Eq. 10 is $u(x,t) = e^{-((x-0.5t)-2)^2}$ However, when executed, the code above will generate the following figure. As can be seen, the numerical solution is not exactly the same as its analytical solution. Overtime, its amplitude is actually decreasing.



1.3. Downwind Method ($c < 0$)

By using finite difference method, we can approximate both u_t and u_x. To approximate u_t, we use the following approximation.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{11}$$

While to approach u_x, we use the following approximation.

$$u_x \approx \frac{u(x+\Delta x,t) - u(x,t)}{\Delta x} \tag{12}$$

Substituting (11) and (12) to (1) yields the following.

$$
\begin{aligned}
u_t &= -cu_x + f(x,t) \\
\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} &\approx -c\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} + f(x,t) \\
u(x,t+\Delta t)+u(x,t) &\approx -\frac{c\Delta t}{\Delta x}\Big[u(x+\Delta x,t)-u(x,t)\Big] + f(x,t)\Delta t \\
u(x,t+\Delta t) &\approx u(x.t) - \frac{c\Delta t}{\Delta x}\Big[u(x+\Delta x,t)-u(x,t)\Big] + f(x,t)\Delta t
\end{aligned}
$$

Consider $\frac{c\Delta t}{\Delta x} = r$, $U_{j,k} = u(x_j,t_k)$, $F_{j,k} = f(x_j,t_k)$, $x_j = j\Delta x$ and $t_k = k\Delta t$, we can have the following shorter terms:

$$
\boxed{U_{j,k+1} = U_{j,k} - r(U_{j+1,k} - U_{j,k}) + F_{j,k}\Delta t} \tag{13}
$$

MATLAB implementation of an upwind method (Eq. 13) is as follows.

```
function next_u_array = downwind(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

for i = 1:N-1
    next_u_array(i) = u_array(i)-c*dt/dx*(u_array(i+1)-u_array(i))+f_array(i)*dt;
end

end
```

To increase its efficiency, we can turn the MATLAB code above in to matrix operations as follows.

```
function next_u_array = downwind(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

next_u_array(1:N-1) = u_array(1:N-1) - (c*dt/dx).* ...
                      (u_array(2:N) - u_array(1:N-1)) + f_array(1:N-1).*dt;

end
```

The following lines of code is used to test the upwind function. Here we want to to solve the following first order linear PDE.

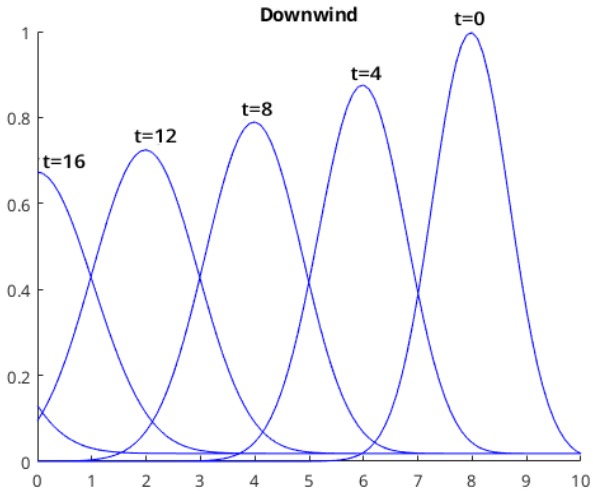
$$
\begin{aligned}
u_t - 0.5u_x &= 0 \\
u(x,0) &= e^{-(x-8)^2} \\
0 \leqslant x &\leqslant 10
\end{aligned} \tag{14}
$$

The length is divided into 100 segments ($dx = 0.1$) and the time is divided into 400 segments ($dt = 0.05$).

```
L = 10;
dx = 0.1;
x = 0:dx:L;

dt = 0.05;
T = 20;
t = 0:dt:T;
```

```
8
9  c = -0.5; % Downwind
10
11 u = zeros(length(x), 1);
12 f = zeros(length(x), 1);
13
14 for k = 1:length(x)
15     u(k) = exp(-(x(k)-8).^2);
16 end
17
18 figure;
19 h = plot(0,0);
20 title('Downwind')
21 ylim([0 1]);
22
23 for k = 1:length(t)
24     u = downwind(u,f, c, dt, dx);
25     set(h,'XData',x,'Ydata',u);
26     drawnow;
27 end
```

The analytical solution of Eq. 14 is $u(x,t) = e^{-((x+0.5t)-8)^2}$ However, when executed, the code above will generate the following figure. As can be seen, the numerical solution is not exactly the same as its analytical solution. Overtime, its amplitude is actually decreasing.



1.4. The Lax Method

The Lax method is based on the FTCS method. However, $u(x,t)$ is replaced by an average of its two neighbors. To approximate u_t, we use the following approximation.

$$u_t \approx \frac{u(x,t+\Delta t) - \boxed{u(x,t)}}{\Delta t} \tag{15}$$

$$u_t \approx \frac{u(x,t+\Delta t) - \left[\frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2}\right]}{\Delta t}$$

While to approach u_x, we use the following approximation.

$$u_x \approx \frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x} \tag{16}$$

Substituting (3) and (4) to (1) yields the following.

$$\begin{aligned}
u_t &\approx -cu_x + f(x,t) \\
\frac{u(x,t+\Delta t) - \left[\frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2}\right]}{\Delta t} &\approx -c\frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x} + f(x,t) \\
u(x,t+\Delta t) - \left[\frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2}\right] &\approx -\frac{c\Delta t}{2\Delta x}\left[u(x+\Delta x,t)-u(x-\Delta x,t)\right] + f(x,t)\Delta t \\
u(x,t+\Delta t) &\approx \left[\frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2}\right] - \\
&\quad \frac{c\Delta t}{2\Delta x}\left[u(x+\Delta x,t)-u(x-\Delta x,t)\right] + f(x,t)\Delta t
\end{aligned} \tag{17}$$

Consider $r = \frac{c\Delta t}{\Delta x}$, $U_{j,k} = u(x_j,t_k)$, $F_{j,k} = f(x_j,t_k)$, $x_j = j\Delta x$ and $t_k = k\Delta t$, we can have the following shorter terms:

$$\boxed{U_{j,k+1} = \frac{U_{j+1,k}+U_{j-1,k}}{2} - \frac{r}{2}(U_{j+1,k}-U_{j,k}) + F_{j,k}\Delta t} \tag{18}$$

The Lax method can handle both upwind and downwind direction. There is no need to check the sign of c as in the previous methods. However, we still have the same issue as in previous methods since the resulting amplitude is also decreasing over time. MATLAB implementation of the Lax method is as follows.

```
1  function next_u_array = lax(u_array, f_array, c, dt, dx)
2
3  N = length(u_array);
4  next_u_array = u_array;
5
6  r = c*dt/dx;
7
8  for i = 2:N-1 % i = 1 and i = N are untouched !!!
9      next_u_array(i) = 0.5*(u_array(i+1)+u_array(i-1)) - ...
10                       0.5*r*(u_array(i+1)-u_array(i-1)) + ...
11                       f_array(i)*dt;
12 end
13
14 % fill up the two ends by using upwind or downwind method
15 if c > 0
16     next_u_array(N) = u_array(N) - r* ...
17                      (u_array(N)- u_array(N-1)) + f_array(N).*dt;
18
19 elseif c < 0
20     next_u_array(1) = u_array(1) - r* ...
21                       (u_array(2) - u_array(1)) + f_array(1).*dt;
22
23 end
```

There is one issue with the Lax method. The Lax method leaves the two end segments untouched. This can be concluded from the for loop statement that starts from 2 to N-1 (line 8). To address this issue, we can apply the upwind and downwind method for the two ends only.

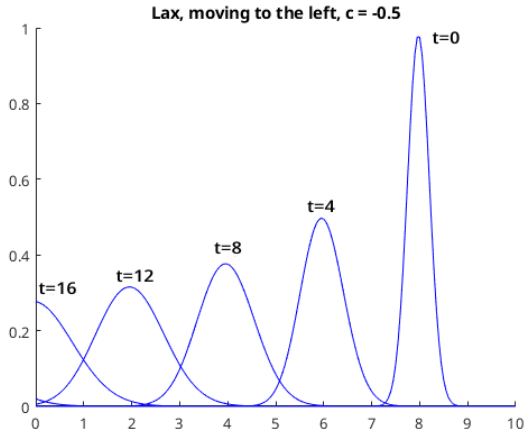
To increase its efficiency, we can turn the MATLAB code above in to matrix operations as follows.

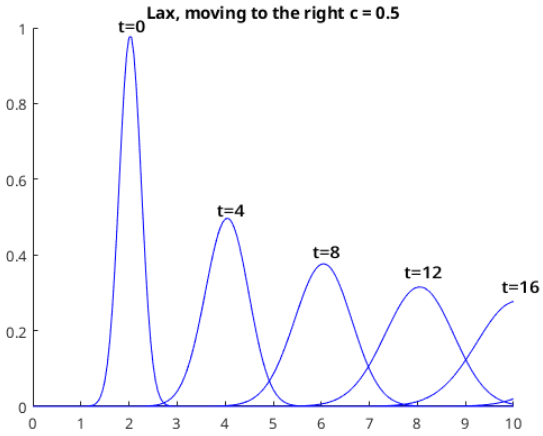
```
function next_u_array = lax(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

r = c*dt/dx;

next_u_array(2:N-1) = 0.5.*(u_array(3:N)+(u_array(1:N-2))) - ...
                      (0.5*r).* (u_array(3:N)- u_array(1:N-2)) + ...
                      f_array(2:N-1).*dt;


% fill up the two ends by using upwind or downwind method
if c > 0
    next_u_array(N) = u_array(N) - r* ...
                      (u_array(N)- u_array(N-1)) + f_array(N).*dt;

elseif c < 0
    next_u_array(1) = u_array(1) - r* ...
                      (u_array(2) - u_array(1)) + f_array(1).*dt;

end
```

Next, we test the code above for both upwind and downwind direction. For upwind direction, we use the same system as in Eq. 10. As for the downwind direction, we use the same system as in Eq. 14. For both methods, the length is divided into 100 segments ($dx = 0.1$) and the time is divided into 400 segments ($dt = 0.05$).

```
L = 10;
dx = 0.05;
x = 0:dx:L;

dt = 0.05;
T = 20;
t = 0:dt:T;

u = zeros(length(x), 1);
f = zeros(length(x), 1);

%% ---------------------------------------------------------------
c = 0.5; % moving to right

for k = 1:length(x)
    u(k) = exp(-10*(x(k)-2).^2);
end

figure;
h1 = plot(0,0);
title('Lax, moving to the right c = 0.5')
ylim([0 1]);

for k = 1:length(t)
    u = lax(u,f, c, dt, dx);
    set(h1,'XData',x,'Ydata',u);
    drawnow;
end

%% ---------------------------------------------------------------
```

```
c = -0.5; % moving to left

for k = 1:length(x)
    u(k) = exp(-10*(x(k)-8).^2);
end

figure;
h2 = plot(0,0);
title('Lax, moving to the left, c = -0.5')
ylim([0 1]);

for k = 1:length(t)
    u = lax(u,f, c, dt, dx);
    set(h2,'XData',x,'Ydata',u);
    drawnow;
end
```





1.5. The Lax-Wendroff Method[6]

We start from the Taylor series definition of $f(x+\Delta x)$:

$$f(x+\Delta x) = f(x)+\Delta x f'(x)+\frac{\Delta x^2}{2!}f''(x)+\frac{\Delta x^3}{3!}f'''(x)+... \tag{19}$$

Thus, taking only up to the second order, we can write the following equations.

$$u(x+\Delta x,t) = u(x,t)+\Delta x u_x+\frac{\Delta x^2}{2}u_{xx} \tag{20}$$

The advection equation can then be approximated as follows

$$\begin{aligned}
u_t &\approx -cu_x+f(x,t)\\
\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} &\approx -c\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}+f(x,t)\\
u(x,t+\Delta t) &\approx u(x,t)-c\frac{\Delta t}{\Delta x}[u(x+\Delta x,t)-u(x,t)]+f(x,t)\Delta t\\
u(x,t+\Delta t) &\approx u(x,t)-c\frac{\Delta t}{\Delta x}\left[\cancel{u(x,t)}+\Delta x u_x+\frac{\Delta x^2}{2}u_{xx}-\cancel{u(x,t)}\right]+f(x,t)\Delta t\\
u(x,t+\Delta t) &\approx u(x,t)-c\Delta t u_x-c\frac{\Delta t\Delta x}{2}u_{xx}+f(x,t)\Delta t
\end{aligned} \tag{21}$$

Since c is the propagation speed of $u(x,t)=u_0(x-ct)$, we can then take $\Delta x=-c\Delta t$. As the result, now we have

the following equation.

$$\boxed{u(x,t+\Delta t) =\approx u(x,t)-c\Delta t u_x+c^2\frac{\Delta t^2}{2}u_{xx}+f(x,t)\Delta t} \tag{22}$$

Next, we need to approximate u_x and u_{xx}. For u_x, we use central difference method as follows.

$$u_x \approx \frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x} \tag{23}$$

As for u_{xx}, we do not use the central difference method, instead, we take $u_x \approx \frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}$. Therefore, we can express u_{xx} as follows.

$$\begin{aligned} u_{xx} &\approx \frac{u_x(x+\Delta x,t)-u_x(x,t)}{\Delta x} \\ &\approx \frac{\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}-\frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}}{\Delta x} \\ &\approx \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{(\Delta x)^2} \end{aligned} \tag{24}$$

Therefore, by substituting Eq. 23 and Eq. 24 to Eq. 22, we can get the following equations.

$$\begin{aligned} u(x,t+\Delta t) &\approx u(x,t)-c\Delta t u_x+c^2\frac{\Delta t^2}{2}u_{xx}+f(x,t)\Delta t \\ &\approx u(x,t)-c\Delta t\left[\frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x}\right]+ \\ &\qquad c^2\frac{\Delta t^2}{2}\left[\frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{\Delta x^2}\right]+f(x,t)\Delta t \\ &\approx u(x,t)-\frac{c\Delta t}{2\Delta x}[u(x+\Delta x,t)-u(x-\Delta x,t)]+ \\ &\qquad \frac{c^2\Delta t^2}{2\Delta x^2}[u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)]+f(x,t)\Delta t \end{aligned}$$

Consider $r=\frac{c\Delta t}{\Delta x}$, $U_{j,k}=u(x_j,t_k)$, $F_{j,k}=f(x_j,t_k)$, $x_j=j\Delta x$ and $t_k=k\Delta t$, we can have the following shorter terms:

$$\boxed{U_{j,k+1}=U_{j,k}-\frac{r}{2}(U_{j+1,k}-U_{j-1,k})+\frac{r^2}{2}(U_{j+1,k}-2U_{j,k}+U_{j-1,k})+F_{j,k}\Delta t} \tag{25}$$

MATLAB implementation of Eq. 25 is as follows.

```
function next_u_array = lax_wendroff(u_array, f_array, c, dt, dx)

N = length(u_array);
next_u_array = u_array;

r = c*dt/dx;

for i = 2:N-1
    next_u_array(i) = u_array(i)-0.5*r*(u_array(i+1)-u_array(i-1)) ...
                      + 0.5*r^2 * (u_array(i+1)-2*u_array(i)+u_array(i-1)) ...
                      + f_array(i)*dt;
end
```

```
13
14 % fill up the two ends by using upwind or downwind method
15 if c > 0
16     next_u_array(N) = u_array(N) - r* ...
17                       (u_array(N)- u_array(N-1)) + f_array(N)*dt;
18
19 elseif c < 0
20     next_u_array(1) = u_array(1) - r* ...
21                         (u_array(2) - u_array(1)) + f_array(1)*dt;
22
23 end
```

Here, we face the same issue as in the Lax method. The Lax-Wendroff method also leaves the two end segments untouched. This can be seen clearly from the for statement that starts from 2 to N-1 (line 8). To address this issue, we then apply upwind and downwind method for the two ends only.

To increase its efficiency, we can turn the MATLAB code above in to matrix operations as follows.

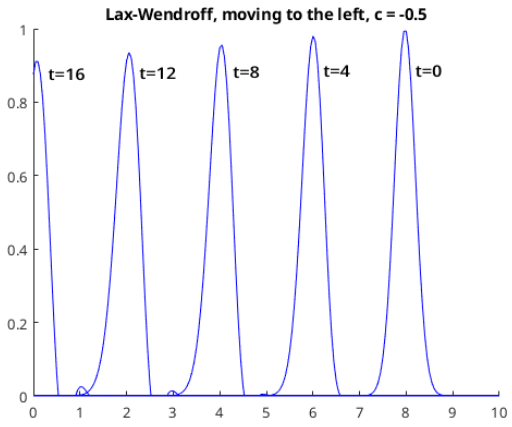
```
1  function next_u_array = lax_wendroff(u_array, f_array, c, dt, dx)
2
3  N = length(u_array);
4  next_u_array = u_array;
5
6  r = c*dt/dx;
7
8  next_u_array(2:N-1) = u_array(2:N-1)-0.5*r*(u_array(3:N)-u_array(1:N-2)) ...
9                        + 0.5*r^2 * (u_array(3:N)-2*u_array(2:N-1)+u_array(1:N-2)) ...
10                       + f_array(2:N-1)*dt;
11
12 % fill up the two ends by using upwind or downwind method
13 if c > 0
14     next_u_array(N) = u_array(N) - r* ...
15                       (u_array(N)- u_array(N-1)) + f_array(N)*dt;
16
17 elseif c < 0
18     next_u_array(1) = u_array(1) - r* ...
19                         (u_array(2) - u_array(1)) + f_array(1)*dt;
20
21 end
```

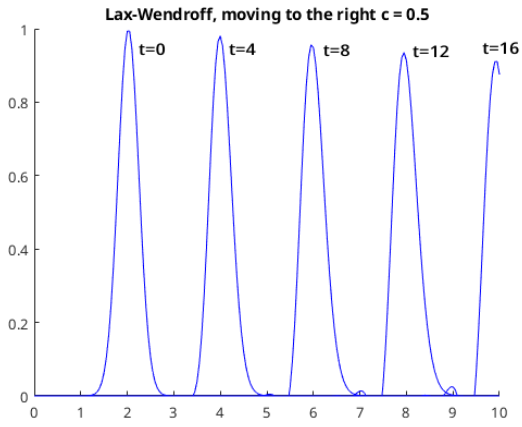
Next, we test the implementations above for both upwind and downwind case. For the upwind case, we will use the same system as in Eq. 10. As for the downwind direction, we will also use the same system as in Eq. 14. We will create 200 segments length-wise $(dx = 0.05)$ and 200 segments time-wise $(dt = 0.05)$.

```
1  L = 10;
2  dx = 0.05;
3  x = 0:dx:L;
4
5  dt = 0.05;
6  T = 20;
7  t = 0:dt:T;
8
9  u = zeros(length(x), 1);
10 f = zeros(length(x), 1);
11
12 %% -------------------------------------------------------------
```

```
c = 0.5; % moving to right

for k = 1:length(x)
    u(k) = exp(-10*(x(k)-2).^2);
end

h = figure;
h1 = plot(0,0);
title('Lax-Wendroff, moving to the right c = 0.5')
ylim([0 1]);

for k = 1:length(t)
    u = lax_wendroff(u,f, c, dt, dx);
    set(h1,'XData',x,'Ydata',u);
    drawnow
end

%% ------------------------------------------------------------
c = -0.5; % moving to left

for k = 1:length(x)
    u(k) = exp(-10*(x(k)-8).^2);
end

h = figure;
h1 = plot(0,0);
title('Lax-Wendroff, moving to the left, c = -0.5')
ylim([0 1]);

for k = 1:length(t)
    u = lax_wendroff(u,f, c, dt, dx);
    set(h1,'XData',x,'Ydata',u);
    drawnow
end
```

When executed, the code above will generate the following figures. As shown by the two figures below, Lax-Wendroff method does provide more stable solutions. Even tough the amplitude is still decreasing, the decrease in the amplitude is much smaller than other methods that have been previously discussed.





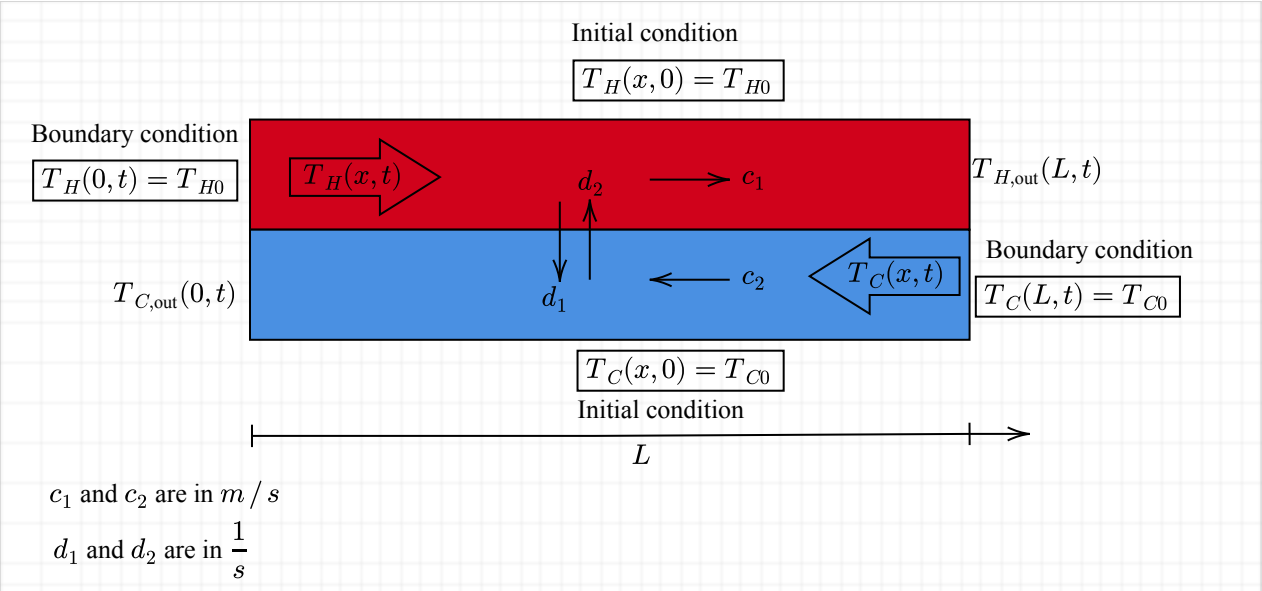
1.6. Cross Current Heat Exchanger as An Example

A heat exchanger is model with two or more interacting non-homogenous PDEs. The simplest model involves two

PDEs only: the hot flow and the cold flow. Yet another simple method, three PDEs are involved: the hot flow, the cold flow, and the wall between them.

1.6.1. Two-Equation Model

Below is a diagram of a simple heat-exchanger. Hot fluid flows from left to right and cold fluid flows from left to right. Source temperature of the hot flow is locked at T_{H0}. While source temperature of the cold flow is locked at T_{C0}. Those two temperatures never change.



The system model of a heat exchanger based on the figure above can be described as follows.

$$\frac{\partial}{\partial t}T_H(x,t) = -c_1\frac{\partial}{\partial x}T_H(x,t) - d_1(T_H(x,t) - T_C(x,t))$$
$$\frac{\partial}{\partial t}T_C(x,t) = c_2\frac{\partial}{\partial x}T_C(x,t) + d_2(T_H(x,t) - T_C(x,t))$$

where:

- d_1 and d_2 describe convective heat transfer between the the two flows.
- c_1 and c_2 describes advective heat transfer inside the the current fluid. This is analogous to the actual movement of the fluids since the heat moves together with the fluids.

For the simulation, we will set the parameters according to[7]. First, we will solve the system above by using the upwind and downwind method, Next, we will use the Lax-Wendroff method and compare the results.

```
L = 1;
dx = 0.2;
x = 0:dx:L;

dt = 0.01;
T = 20;
t = 0:dt:T;

% Define the parameters
c1 = 0.001711593407;
c2 = -0.01785371429;
```

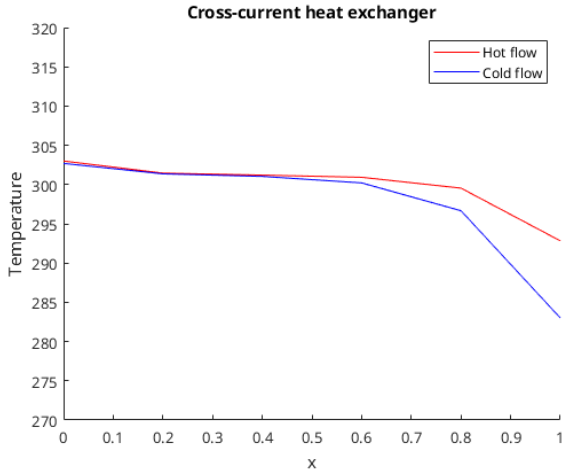
[7] PDE Observer Design for Counter-Current Heat Flows in a Heat-Exchange by F. Zobiri, et. al., published at IFAC-PapersOnLine

```
d1 = -0.03802197802;
d2 = 0.3954285714;

% ========================================================================
% Upwind-downwind method
% ========================================================================
TH = zeros(length(x), 1);
TC = zeros(length(x), 1);
f = zeros(length(x), 1);

% Intial condition
TH(1:length(x)) = 273+30;
TC(1:length(x)) = 273+10;

figure;
hold on
h1 = plot(0,0,'r');
h2 = plot(0,0,'b');
title('Cross-current heat exchanger');
ylim([270 320]);
xlabel('x');
ylabel('Temperature');
legend('Hot flow','Cold flow');

for k = 1 : length(t)
    set(h1,'XData',x,'Ydata',TH);
    set(h2,'XData',x,'Ydata',TC);
    drawnow;

    f = d1.*(TH-TC);
    TH = upwind(TH, f, c1, dt, dx);
    f = d2.*(TH-TC);
    TC = downwind(TC, f, c2, dt, dx);
end

% ========================================================================
% Lax-Wnderoff
% ========================================================================
TH = zeros(length(x), 1);
TC = zeros(length(x), 1);
f = zeros(length(x), 1);

% Intial conditoin
TH(1:length(x)) = 273+30;
TC(1:length(x)) = 273+10;

figure;
hold on
h3 = plot(0,0,'r');
h4 = plot(0,0,'b');
title('Cross-current heat exchanger');
ylim([270 320]);
xlabel('x');
ylabel('Temperature');
legend('Hot flow','Cold flow');

for k = 1:length(t)
    set(h3,'XData',x,'Ydata',TH);
    set(h4,'XData',x,'Ydata',TC);
    drawnow;

```

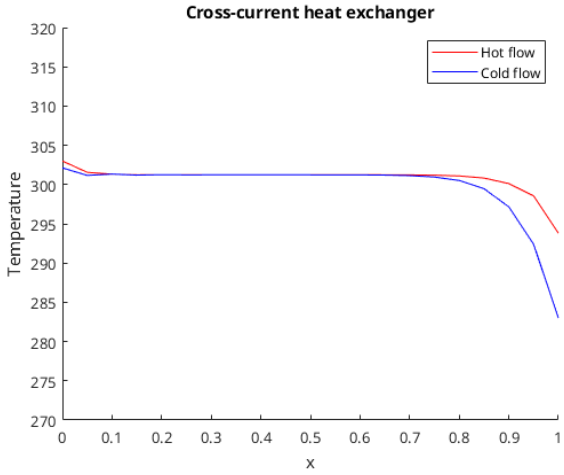
```
73      f = d1.*(TH-TC);
74      TH = lax_wendroff(TH, f, c1, dt, dx);
75      f = d2.*(TH-TC);
76      TC = lax_wendroff(TC, f, c2, dt, dx);
77  end
```

From line 41 to 44, we can see that the hot flow uses the upwind method since the flow moves from left to right. As for the cold flow, we use the downwind method since the flow moves from right to left. As for the Lax-Wendroff method, there is only one function that we need to call (line 73 to 76). However, we need to call that function twice, one for the cold flow, another one is for the hot flow. When executed, the provided code above will generate animations of the temperature evolutions for both methods.

The two figures below show the temperature distribution of the modeled heat exchanger after 20 seconds for two different methods. From visual observations, we can see that Lax-Wendroff method generates flat curve in its middle area. Theoretically, this makes more sense when compared to the results acquired from the upwind-downwind method. However, if we think for a practical perspective, upwind-and downwind method might be more desirable since a theoretically perfect advection might be impossible in a real world.

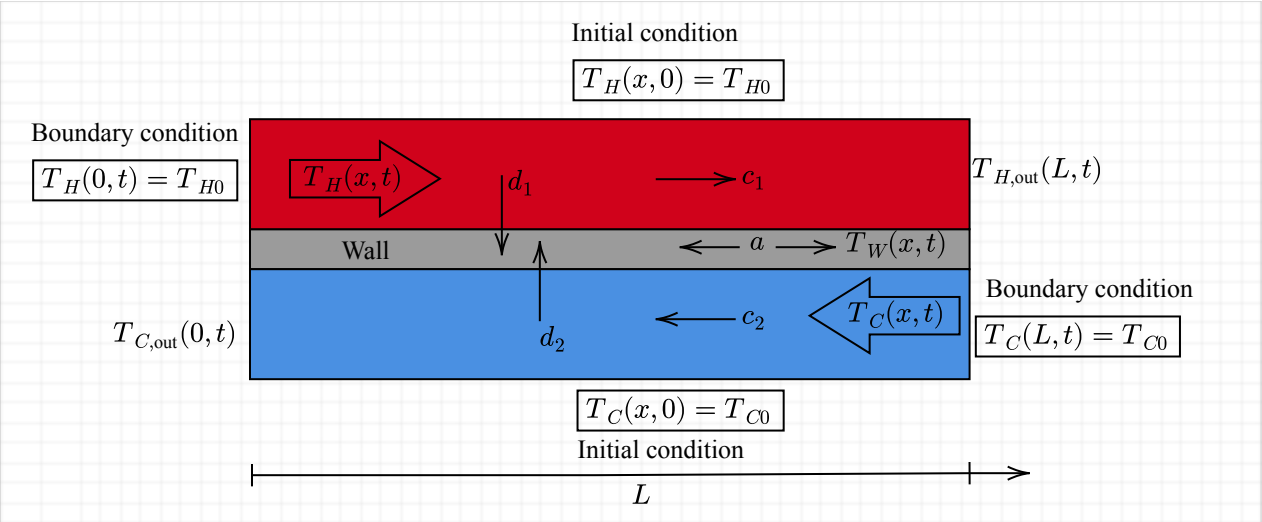


Upwind-downwind method



Lax-Wendroff method

1.6.2. Three-Equation Model



In three-equation model, we consider the wall between the hot flow and the cold flow. In general, we can summarize how the heat are being exchanged as follows.

- d_1 and d_2 describe convective heat transfer between the fluid and the tube wall.
- c_1 and c_2 describes advective heat transfer inside the the current fluid. This is analogous to the movement of the fluid since the heat moves together with the fluid.
- a describes diffusive/conductive heat transfer inside the tube wall.

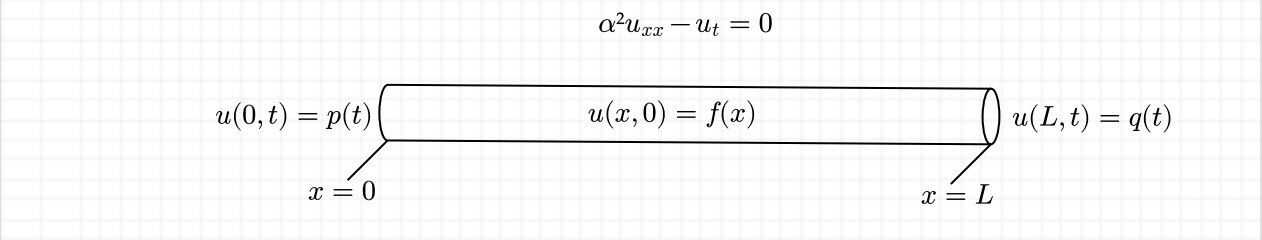
Therefore, the three-equation model of a cross-current heat exchanger can be expressed as follows.

$$\begin{aligned}
\frac{\partial}{\partial t}T_H(x,t) &= -c_1\frac{\partial}{\partial x}T_H(x,t) - d_1(T_H(x,t) - T_W(x,t)) \\
\frac{\partial}{\partial t}T_C(x,t) &= c_2\frac{\partial}{\partial x}T_C(x,t) + d_2(T_W(x,t) - T_C(x,t)) \\
\frac{\partial}{\partial t}T_w(x,t) &= a\frac{\partial^2}{\partial x^2}T_W(x,t) + d_1(T_H(x,t) - T_W(x,t)) - d_2(T_W(x,t) - T_C(x,t))
\end{aligned} \tag{26}$$

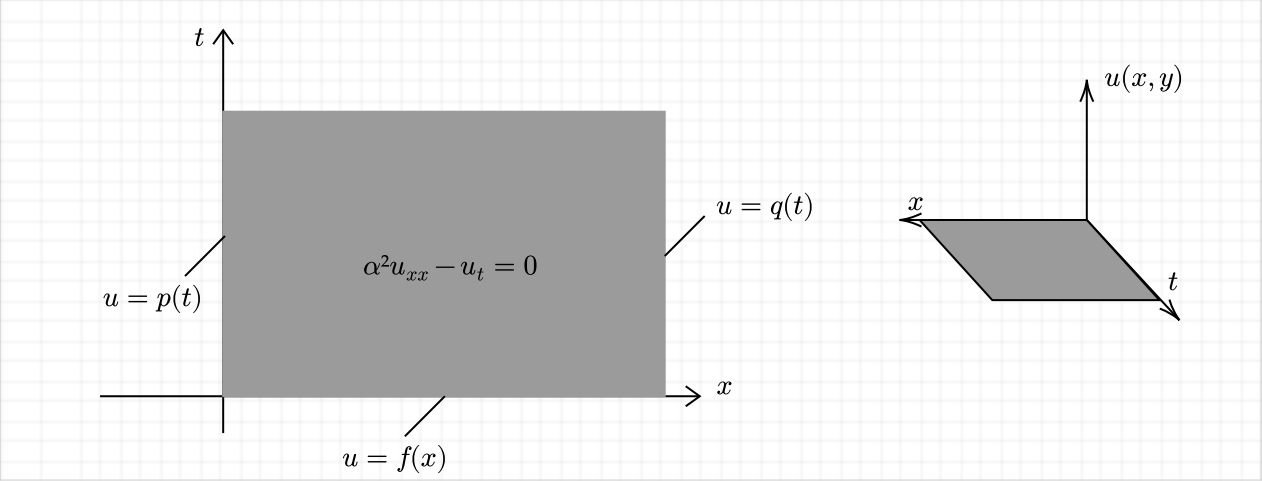
However, the third equation is not an advection equation. It is actually a non-homogenous diffusion equation in a 1-dimensional rod, which we will discuss in the next section. Therefore, we will solve the system above in the next section.

2. Case 2: 1D-Rod[8]

Temperature distribution (u) in a one-dimensional rod over a certain distance (x) and time period (t). Certain conditions (a, b) are applied at both ends. These conditions remain true forever.



Top view of the temperature distribution along the rod over a certain period of time can be presented in a 2D-Cartesian plane as follows. This kind of plot typically done with color plot where every different color represents different temperature.



Given a temperature function of a 1-dimensional rod $u(x,t)$ where u is the temperature at location x and at time t. α is the diffusivity coefficient.

(27)

2.1. Explicit Method

With finite difference method, we can approach u_t and u_{xx}. To approximate u_t, we use the following approximation.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{28}$$

$$u_x \approx \frac{u(x+\Delta x,t) - u(x,t)}{\Delta x} \tag{29}$$

While to approach u_x, we use the following approximation.

$$\begin{aligned} u_{xx} &\approx \frac{u_x(x+\Delta x,t) - u_x(x,t)}{\Delta x} \\ &\approx \frac{\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} - \frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}}{\Delta x} \end{aligned}$$

$$\approx \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{(\Delta x)^2} \tag{30}$$

Substituting (4) to (1):, we can get the following equations.

$$\begin{aligned}
u_t &\approx \alpha^2 u_{xx} \\
\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} &\approx \alpha^2 \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{(\Delta x)^2} \\
u(x,t+\Delta t)-u(x,t) &\approx \frac{\alpha^2\Delta t}{(\Delta x)^2}(u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)) \\
u(x,t+\Delta t) &\approx \frac{\alpha^2\Delta t}{(\Delta x)^2}u(x+\Delta x,t)+\left(1-2\frac{\alpha^2\Delta t}{(\Delta x)^2}\right)u(x,t)+\frac{\alpha^2\Delta t}{(\Delta x)^2}u(x-\Delta x,t) \\
u(x,t+\Delta t) &\approx ru(x+\Delta x,t)+(1-2r)u(x,t)+ru(x-\Delta x,t),\ r=\frac{\alpha^2\Delta t}{(\Delta x)^2}
\end{aligned} \tag{31}$$

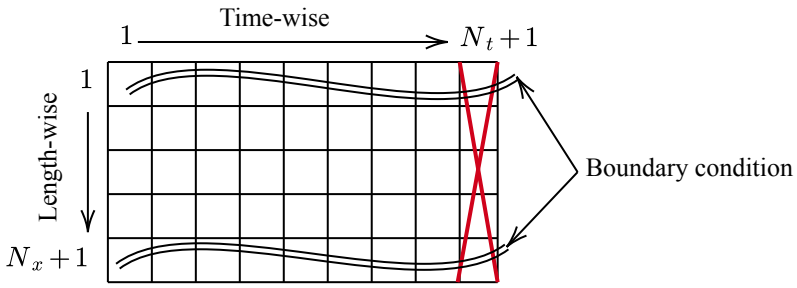
Consider $U_{jk}=u(x_j,t_k)$, $x_j=j\Delta x$, and $t_k=k\Delta t$, we can have the following shorter terms.

$$U_{j,k+1}=u(x,t+\Delta t) \tag{32}$$
$$U_{j-1,k}=u(x-\Delta x,t) \tag{33}$$
$$U_{j,k}=u(x,t) \tag{34}$$

Therefore, additionally taking $r=\alpha^2\frac{\Delta t}{(\Delta x)^2}$, we can shorten the equation (21) into as follows.

$$\boxed{U_{j,k+1}=rU_{j-1,k}+(1-2r)U_{j,k}+rU_{j+1,k}} \tag{35}$$

MATLAB implementation of Eq. 34 is listed below. Eq. 34 can bee seen in line 30. The MATLAB function: diffusion_1d will generate matrix U with a dimension of $(N_x+1)\times Nt$.




```
% alpha^2 * u_xx = u_tt
% Divide 0<=x<=L into Nx segments
% Divide 0<=t<=tF into Nt segments
% p(t) is the left boundary conditoin
% q(t) is the right boundary condition
% f(x) is the initial condition

function U = diffusion_1d(alpha, p, q, f, L, Nx, Nt, tF)

Delta_x = L/Nx;
Delta_t = tF/Nt;

% Prepare the matrix u
U = zeros(Nx+1,Nt+1);

% Apply the initial condition U(x,0)=f(x)
```

```
for j = 1 : Nx+1
  U(j,1) = f((j-1)*Delta_x);
end

r = alpha^2*Delta_t/Delta_x^2;

for k = 1 : Nt     % time-wise

  % Keep the boundary condition
  U(1,k) = p((k-1)*Delta_t);
  U(Nx+1,k) = q((k-1)*Delta_t);

  for j = 2 : Nx   % length-wise
    U(j,k+1) = r*U(j-1,k) + (1-2*r) * U(j,k) + r*U(j+1,k);
  end

end

U = U(:,1:Nt); % the last column is untouched, remove it

end
```

The vectorized version of the code above is as follows. In line 20 of the code below, there is still one for statement. We can not get rid of this for statement as in the time-wise loop, the current iteration depends on the previous iteration.

```
function U = diffusion_1d(alpha, p, q, f, L, Nx, Nt, tF)

Delta_x = L/Nx;
Delta_t = tF/Nt;

% Prepare the matrix u
U = zeros(Nx+1,Nt+1);

% Apply the initial condition U(x,0)=f(x)
j = 1:Nx+1;
U(j,1) = f((j-1)*Delta_x);

% Keep the boundary condition
k = 1:Nt;
U(1,k) = p((k-1)*Delta_t);
U(Nx+1,k) = q((k-1)*Delta_t);

r = alpha^2*Delta_t/Delta_x^2;

for k = 1:Nt      % time-wise
    j = 2 : Nx;   % length-wise
    U(j,k+1) = r*U(j-1,k) + (1-2*r) * U(j,k) + r*U(j+1,k);
end

U = U(:,1:Nt); % the last column is untouched, remove it

end
```

As an example, we will solve the following problem using the MATLAB code above.

$$u_{xx} - u_t = 0,\ 0 \leqslant x \leqslant 1,\ 0 \leqslant t \leqslant 1 \tag{36}$$

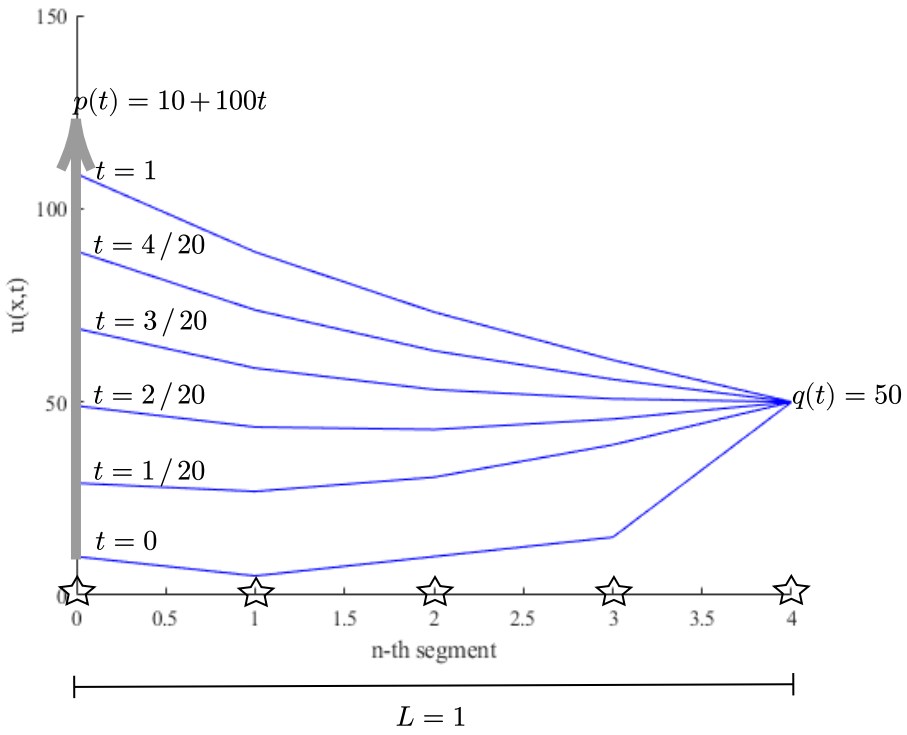
The boundary and initial conditions are as follows.

$$u(0,t) = p(t) = 10 + 100t$$

$$u(L,t) = q(t) = 50 \\ u(x,0) = f(x) = 20x \tag{37}$$

Implementation of the problem given by Eq. 35 is as follows. The actual implementation is from line 1 to line 11. The rest of the code is only for visualization. When executed, it will display how the temperature distribution along the rod evolves over time.

```
1  alpha = 1;
2  L = 1;
3  tF = 1;
4  Nx = 4;
5  Nt = 100;
6
7  p = @(t) 10+100*t;
8  q = @(t) 50;
9  f = @(x) 20*x;
10
11 U = diffusion_1d(alpha, p, q, f, L, Nx, Nt, tF);
12
13 % Plot and animation
14 figure
15 hold on
16 h = plot(0,0);
17 ylim([0 150])
18 xlabel('x')
19 ylabel('u(x,t)')
20 x = 0 : Nx;
21 for k = 1 : Nt
22     set(h, 'XData', x, 'YData', U(:,k))
23     drawnow;
24 end
```



2.2. Implicit Method

The principal idea of an implicit method is to calculate u_{xx} not only at the current time, but also at the time one step ahead.

$$u_{xx}(x,t) = (1-\theta)u_{xx}(x,t) + \theta u_{xx}(x,t+\Delta t),\ 0 \leqslant \theta \leqslant 1 \tag{38}$$

where θ acts as a weighting factor. When $\theta = 0$, (10) becomes similar to (4) and what we have here is an explicit method that is similar to Forward Euler Method. On the other hand, when $\theta = 1$, what we have we here is a very common explicit method that is similar to Backward Euler method. When $\theta = 0.5$, the method is called Crank-Nicolson method. As in the Greenberg's book, we will focus the derivation of the Crank-Nicolson method.

$$\begin{aligned}
u_{xx} &= \frac{1}{2}\frac{u_x(x+\Delta x,t)-u_x(x,t)}{\Delta x} + \frac{1}{2}\frac{u_x(x+\Delta x,t+\Delta t)-u_x(x,t+\Delta t)}{\Delta x} \\
&= \frac{\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} - \frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}}{2\Delta x} + \frac{\frac{u(x+\Delta x,t+\Delta t)-u(x,t+\Delta t)}{\Delta x} - \frac{u(x,t+\Delta t)-u(x-\Delta x,t+\Delta t)}{\Delta x}}{2\Delta x} \\
&= \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{2(\Delta x)^2} \\
&\qquad + \frac{u(x+\Delta x,t+\Delta t)-2u(x,t+\Delta t)+u(x-\Delta x,t+\Delta t)}{2(\Delta x)^2} \\
&= \frac{U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}}{2(\Delta x)^2}
\end{aligned} \tag{39}$$

Substituting (29) back to $\alpha^2 u_{xx} - u_t = 0$ gives us following equation.

$$\begin{aligned}
u_t &= \alpha^2 u_{xx} \\
\frac{U_{j,k+1}-U_{j,k}}{\Delta t} &= \alpha^2 \frac{U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}}{2(\Delta x)^2}
\end{aligned} \tag{40}$$

Since $r = \alpha^2 \dfrac{\Delta t}{(\Delta x)^2}$, (12) can then be simplified as:

$$\begin{aligned}
u_t &= \alpha^2 u_{xx} \\
U_{j,k+1}-U_{j,k} &= \frac{r}{2}(U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}) \\
U_{j,k+1}+rU_{j,k+1} &= \frac{r}{2}U_{j+1,k}+(1-r)U_{j,k}+\frac{r}{2}U_{j-1,k}+\frac{r}{2}U_{j+1,k+1}+\frac{r}{2}U_{j-1,k+1} \\
2(1+r)U_{j,k+1} &= rU_{j+1,k}+2(1-r)U_{j,k}+rU_{j-1,k}+rU_{j+1,k+1}+rU_{j-1,k+1} \\
-rU_{j-1,k+1}+2(1+r)U_{j,k+1}-rU_{j+1,k+1} &= rU_{j-1,k}+2(1-r)U_{j,k}+rU_{j+1,k}
\end{aligned} \tag{41}$$

Thus, the well-known Crank-Nicolson scheme can be expressed as follows.

$$\boxed{-rU_{j-1,k+1}+2(1+r)U_{j,k+1}-rU_{j+1,k+1} = rU_{j-1,k}+2(1-r)U_{j,k}+rU_{j+1,k}} \tag{42}$$

In matrix form:

$$\underbrace{\begin{bmatrix}
-r & 2(1+r) & -r & & \cdots & 0 \\
0 & -r & 2(1+r) & -r & & \vdots \\
 & & -r & 2(1+r) & -r & \\
 & & & \ddots & & \\
\vdots & & & & -r & 2(1+r) \\
0 & \cdots & & & 0 & -r
\end{bmatrix}}_{A}
\underbrace{\begin{bmatrix}
p((k+1)\Delta t) \\
U_{1,k+1} \\
U_{2,k+1} \\
\vdots \\
U_{N-2,k+1} \\
q((k+1)\Delta t)
\end{bmatrix}}_{U} =$$

$$\underbrace{\begin{bmatrix} r & 2(1-r) & r & & \cdots & 0 \\ 0 & r & 2(1-r) & r & & \vdots \\ & & r & 2(1-r) & r & \\ & & & \ddots & & \\ \vdots & & & & r & 2(1-r) \\ 0 & \cdots & & & 0 & r \end{bmatrix} \begin{bmatrix} p(k\Delta t) \\ U_{1,k} \\ U_{2,k} \\ \vdots \\ U_{N-2,k} \\ q(k\Delta t) \end{bmatrix}}_{B} \tag{43}$$

Red means the values are known. In (15), we can notice that some known values reside in U. We must remove these known values from U and put them to the right side. The goal is to have a perfect $AU = B$ system where all the unknowns are all in U.

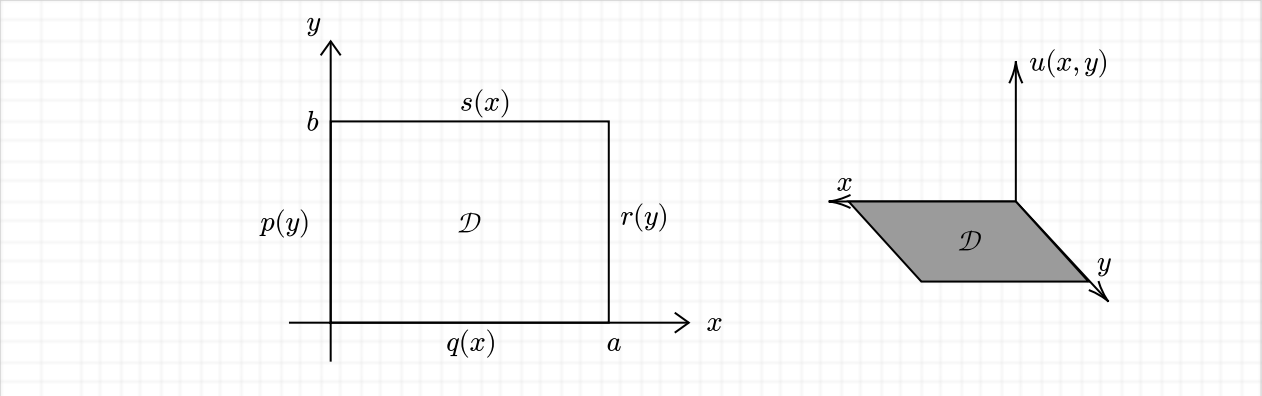
$$\underbrace{\begin{bmatrix} 2(1+r) & -r & & \cdots & 0 \\ -r & 2(1+r) & -r & & \vdots \\ & & \ddots & & \\ & & -r & 2(1+r) & -r \\ \cdots & & & -r & 2(1+r) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_{1,k+1} \\ U_{2,k+1} \\ \vdots \\ U_{N-2,k+1} \\ U_{N-1,k+1} \end{bmatrix}}_{U_{new}} =$$

$$\underbrace{\underbrace{\begin{bmatrix} 2(1-r) & r & & \cdots & \cdots \\ r & 2(1-r) & r & & \\ & \ddots & & & \\ & & r & 2(1-r) & r \\ \cdots & & & r & 2(1-r) \end{bmatrix}}_{b} \underbrace{\begin{bmatrix} U_{1,k} \\ U_{2,k} \\ \vdots \\ U_{N-2,k} \\ U_{N-1,k} \end{bmatrix}}_{U_{now}} + \underbrace{\begin{bmatrix} rp((k+1)\Delta t) + rp(k\Delta t) \\ 0 \\ \vdots \\ 0 \\ rq((k+1)\Delta t) + rq(k\Delta t) \end{bmatrix}}_{c}}_{B=(b\cdot U_{now})+c} \tag{44}$$

Now, we can find $U_{new} = A^{-1}B$. In real implementation, this might not be plausible and we have to perform an iteration-based algorithm to find the optimal U_{new}. In Maple, we can use the U_{new} = LinearSolve(A,B) command. In MATLAB, we can use the popular $U_{new} = A \setminus B$ command. Other alternatives are using the Jacobi method and the Gauss-Siedel method as explained the Greenberg's book.

3. Case 3: Rectangular 2D Plane[9]

3.1. The Five-Point Difference Method



Finite difference method for the system above is as follows.

3.2. Energy Balance Model: Out-of-Plane Heat Transfer for a Thin Plate - Comparison with COMSOL